Understanding C++ Using the AST and the Object Model

Tim Shull

timothyshull@gmail.com

https://github.com/timothyshull/object_model

About me…

- Varied professional/educational background (mathematics, music, financial, music technology)
- Transitioned to software engineering about 2.5 years ago, initially with Ruby, switched to frontend JavaScript
- Started first job as frontend JavaScript engineer, added in server-side JavaScript with Node.js, and then transitioned to DevOps with Python and Ansible for cluster management
- Currently filling in gaps in CS knowledge, focusing on C++
- Learn best through hands-on manipulation of low-level details and building up from there
- Most languages have layers and many have a runtime written in C++ (JVM, V8)
- C and C++ allow direct inspection of the compiled instructions but …

it takes work and the right tools ;)

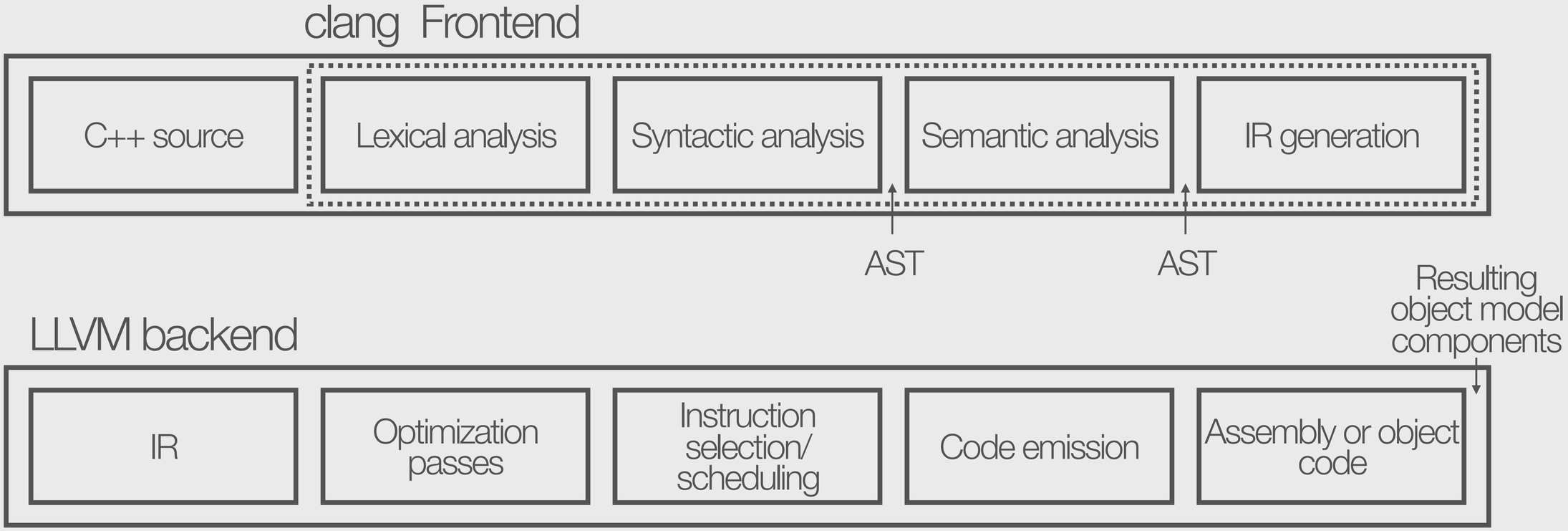
Why this topic?

- Motivated by an effort to gain hands-on knowledge about the inner workings of C++
- The standard provides a wealth of information but hard to parse and retain for practical purposes
- The AST provides a snapshot of the code structure and interpretation before translation to IR, optimization, and subsequent translation to machine instructions
- A good debugger can provide most of the information about the resulting machine instructions
- Debugger requires more effort and specialized knowledge (e.g. understanding of assembly for particular architecture) or debug symbols which can hide the actual instructions in a release executable
- Hopper simplifies disassembly and interaction with compiled code (release or debug)

Why this topic?

- Stanley Lippman's Inside the C++ Object Model was the first resource I came across that really went through these details in a clear and mostly comprehensive way
- A better title for the talk might have been "Understanding C++ by Understanding the Object Model using the AST and Hopper" ;)
- Learning a complex subject like C++ is difficult -
  - Best approached incrementally, similar to practicing a musical instrument
  - Learn from the top down by writing larger projects, similar to learning a large piece of music
  - Improve from the bottom up by honing knowledge of the lower level details, similar to practicing scales

Why this topic?



Initial Background

- Limiting the exercise to macOS, a Unix-based OS built from components of NextSTEP, BSD, and Mach, with many of the core components released as Darwin, an open-source OS
- Mach is a microkernel originally developed at Carnegie Mellon, which is important because we will be looking at Mach-O executables
- Object model - loosely defined as the language features that support object-oriented programming and the underlying mechanisms by which this support is implemented
- An object-oriented language defines (explicitly or implicitly) its object model through the properties exposed by and inherent to objects and the semantics of interaction with those objects through code constructs

Initial Background

- AST (Abstract Syntax Tree) - “tree representation of the abstract syntactic structure of [the] source code” within a given program
- The term parse tree is mostly synonymous with syntax tree, which is a generalized term to refer to a data structure that represents the relationship between program text and the program’s grammatical structure
- AST is a specialization of these terms which includes (often detailed) information about the semantics of the code within the nodes of the data structure

Object Model - from the Standard

The constructs in a C++ program create, destroy, refer to, access, and manipulate objects.

An object is created by a definition, by a new-expression, when implicitly changing the active member of a union, or when a temporary object is created.

An object occupies a region of storage in its period of construction, throughout its lifetime, and in its period of destruction.

The properties of an object are determined when the object is created. An object can have a name. An object has a storage duration which influences its lifetime. An object has a type. Some objects are polymorphic; the implementation generates information associated with each such object that makes it possible to determine that object's type during program execution. For other objects, the interpretation of the values found therein is determined by the type of the expressions used to access them.

Object Model

- Creation
- Destruction
- Access and manipulation
- Storage
- Lifetime
- Temporaries
- Type
- Polymorphism
- Additionally improper usage (exceptions)

Object Model Generally

- Operations
- Requests/messages/methods/communication
- Specification of behavioral semantics
- State
- Encapsulation
- Lifetime
- Behavior and state, e.g. state transitions
- Types/classes
- Inheritance and delegation
- Polymorphism and binding
- Identity, equality, copy (and move)
- Extensibility
- Layout

Stanley Lippman's Outline

- Construction
  - Default
  - Copy
  - Resulting program transformation
  - Member initialization lists
- Data/data members
  - binding
  - layout
  - access
  - inheritance
  - efficiency
  - pointers to data members
- Functions
  - types (e.g. static, non-static, member, non-member, inline, virtual members, pointers to functions/members)
  - function efficiency
- Construction, destruction, and copy
  - with and without inheritance
  - efficiency
- Runtime semantics
  - construction and destruction
  - operators new and delete
  - temporaries
- Templates
- Exception Handling
- Runtime Type Identification

Additional Considerations

- Move semantics
- Memory model (and parallel/concurrent facilities)
- Changes for modern C++

Each component contributes to the runtime…

Runtime

- runtime system vs. runtime environment
- from Wikipedia (https://en.wikipedia.org/wiki/Runtime_system):
  - One (debatable) way to define a runtime system is that any behavior that is not directly the work of a program is runtime system behavior. This definition includes as part of the runtime system things such as putting parameters onto the stack before a function call, the behavior of disk I/O, and parallel execution related behaviors.
  - The runtime system is also the gateway by which a running program interacts with the runtime environment, which contains not only state values that are accessible during program execution, but also active entities that can be interacted with during program execution like disk drives and people, via keyboards.
- runtime environment requires adequate consideration to understand our approach…

Mach-O Executables

- Full picture requires consideration of the runtime architecture
  - The static linker adds the standard entry point function to the executable at build time
  - `fork` creates a process, some flavor of `exec` ultimately calls `execve` in the kernel which will load the executable at the path passed to `execve`
  - The Mach-O header (mach_header) contains load commands which are verified by the kernel when the executable is loaded
  - One of the commands contains the path to the dynamic linker (/usr/bin/dyld), which is then executed by the kernel
  - The dynamic linker loads and binds enough of the symbols from the shared libraries to run the program and then calls the standard entry point function, which calls static initializers for C++ objects, initializes the Objective-C runtime, and then calls main

Mach-O Executable Initialization

- dyld start
- dyld dyldbootstrap::start
- dyld dyld::main
- dyld dyld::initializeMainExecutable
- dyld ImageLoader::runInitializers
- dyld ImageLoader::processInitializers
- dyld ImageLoader::recursiveInitialization
- dyld ImageLoaderMachO::doInitialization
- dyld ImageLoaderMachO::doModInitFunctions
- libSystem.B.dylib libSystem initializer
- libdispatch.dylib libdispatch init
- libdispatch.dylib os object init
- libobjc.A.dylib objc init
- libdyld.dylib dyld objc notify register

- dyld dyld::registerObjCNotifiers
- dyld dyld::notifyBatchPartial
- libobjc.A.dylib map images
- libobjc.A.dylib map images nolock
- libobjc.A.dylib read images
- …

- see https://opensource.apple.com/source/dyld
- dyldInitialization.cpp
- ImageLoaderMachO.cpp

Mach-O Executable Layout

- Header
  - Identifies the file as a Mach-O executable file
  - Contains basic file type information
  - Specifies the target CPU architecture
  - Contains flags that affect the interpretation of the remaining portion of the file
  - See /usr/include/mach-o/loader.h

Mach-O Executable Layout

- Load commands
  - Initial layout of the file in virtual memory
  - Location of the symbol table for dynamic loading and debugging
  - The initial execution state of the main thread of the program
  - The names of the shared libraries containing the executable’s imported symbols

Mach-O Executable Layout

- Segments (specified by load commands)
  - Each section of a segment contains code or data of a particular type
  - Dynamic linker maps segments into the process address space
  - __PAGEZERO - located at memory address 0, no protection rights, size of one full VM page for current architecture but occupies no space in the file
  - __TEXT - executable code and read-only data, read-only protection rights
  - __DATA - writable data segment
  - __OBJC - data used by the Objective-C runtime support library
  - __LINKEDIT - raw data used by the dynamic linker (e.g. symbol, string, and relocation table entries)

Mach-O Executable Layout (e.g.)

- Mach64 Header
- Load Commands
  - LC_SEGMENT_64 (__PAGEZERO)
  - LC_SEGMENT_64 (__TEXT)
  - LC_SEGMENT_64 (__LINKEDIT)
  - LC_DYLD_INFO_ONLY
  - LC_SYMTAB
  - LC_DYSYMTAB
  - LC_LOAD_DYLINKER
  - LC_UUID
  - LC_VERSION_MIN_MACOSX
  - LC_SOURCE_VERSION
  - LC_MAIN
  - LC_LOAD_DYLIB (libc++.1.dylib)
  - LC_LOAD_DYLIB (libSystem.dylib)
  - LC_FUNCTION_STARTS
  - LC_DATA_IN_CODE
- Section64(__TEXT, __text)
  - Assembly
- Section64(__TEXT, __stubs)
  - Symbol stubs
- Section64(__TEXT, __stub_helper)
  - Assembly
- Section64(__TEXT, __unwind_info)
- Section64(__DATA, __nl_symbol_ptr)
  - Non-lazy Symbol Pointers
- Section64(__DATA, __la_symbol_ptr)
  - Lazy Symbol Pointers
- Dynamic Loader Info
  - Binding Info
  - Weak Binding Info
  - Export Info
- Function Starts
- Symbol Table
- Data in Code Entries
- Dynamic Symbol Table
- String Table

Useful Commands

- `nm <executable>`
  - nm displays the name list (symbol table) of each object file in the argument list. If an argument is an archive, a listing for each object file in the archive will be produced.
- `size -x -l -m <executable>`
  - size prints the size of the sections in an object file.
  - -x -l -m - print the sizes, addresses, and offsets of the Mach-O segments and sections as well as the total sizes of the sections in each segment and the total size of the segments in the file in hexadecimal.
- `otool`
  - otool displays specified parts of object files or libraries.
  - `otool -hv <executable>` - displays the Mach-O headers.
  - `otool -lv <executable>` - displays the load commands.
  - `otool -tV <executable>` - disassembles the contents of the (__TEXT,__text) section, displaying the operands symbolically.

Warm-up

```
struct X {
    double x;
    double y;
};

struct Y {
    double v;
    Y *n;
};

int main()
{
    X x1;
    Y y1;

    X x2{};
    Y y2{};

    X x3{1.0, 2.0};
    Y y3{3.0, &y2};
    return 0;
}
```

- Please note;
  - The code examples are intentionally relatively simple to promote the ability to map between the various representations
  - I generally used very short class, function, and method names to reduce noise in the mangled names (I am still not sure that this was the best decision)

Hopper Demo

x86 64-bit Execution Environment

Basic Program Execution Registers

Sixteen 64-bit Registers	General Purpose Registers
Six 16-bit Registers	Segment Registers
64-bits	RFLAGS Register
64-bits	RIP (Instruction Ptr) Register

FPU Registers

Eight 80-bit Registers	Floating-Point Data Registers
16-bits	Control Register
16-bits	Status Register
16-bits	Tag Register
	Opcode Register (11-bits)
64-bits	FPU Instruction Ptr Register
64-bits	FPU Data (Operand) Ptr Register

MMX Registers

Eight 64-bit Registers

Bounds Registers

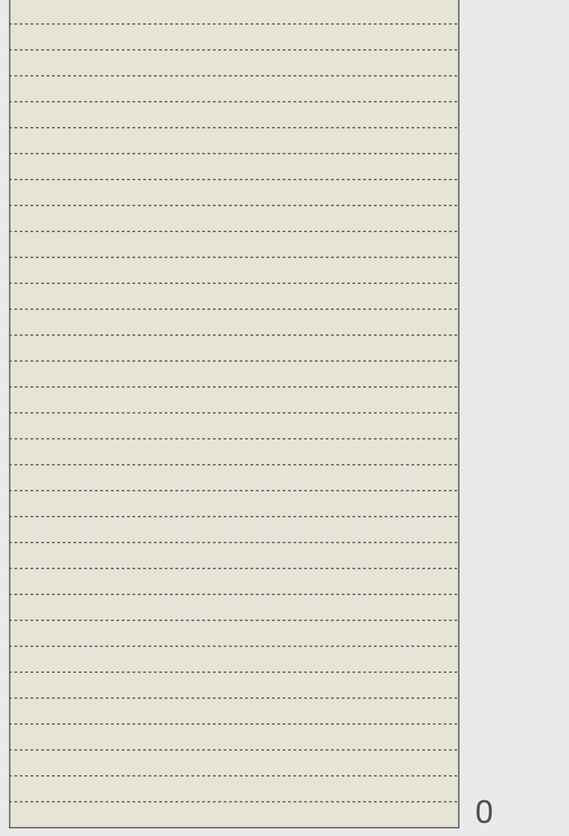
Four 128-bit Registers

BNDCFGU BNDCFGU

XMM Registers

Sixteen 128-bit Registers	XMM Registers
32-bits	MXCSR Register
Sixteen 256-bit Registers	YMM Registers

YMM Registers



Registers

RFLAGS

Program Status and Control

RIP

Instruction Pointer

RAX
RBX
RCX
RDX
RSI
RDI
RBP
RSP
R8
R9
R10
R11
R12
R13
R14
R15

General Purpose Registers

R7 (MM7)
R6 (MM6)
R5 (MM5)
R4 (MM4)
R3 (MM3)
R2 (MM2)
R1 (MM1)
R0 (MM0)

X87 Register Stack (MMX Registers)

X87 Control, Status, and Tags Registers

YMM0/XMM0
YMM1/XMM1
YMM2/XMM2
YMM3/XMM3
YMM4/XMM4
YMM5/XMM5
YMM6/XMM6
YMM7/XMM7
YMM8/XMM8
YMM9/XMM9
YMM10/XMM10
YMM11/XMM11
YMM12/XMM12
YMM13/XMM13
YMM14/XMM14
YMM15/XMM15

AVX/SSE Registers

MXCSR

AVX/SSE Control and Status

x87 FPU


79
64
0
R7
R6
R5
R4
R3
R2
R1
R0
S
Exponent
Significand
Data Registers
Control Register
Status Register
Tag Register
Last Instruction Ptr
Last Data Ptr
Last Instruction Opcode
Special-Purpose Registers

GPR Detail

64-bit, 32-bit, 16-bit, and 8-bit Registers

64	31	15	7 0
RAX	EAX	AX	AL
RBX	EBX	BX	BL
RCX	ECX	CX	CL
RDX	EDX	DX	DL
RSI	ESI	SI	SIL
RDI	EDI	DI	DIL
RBP	EBP	BP	BPL
RSP	ESP	SP	SPL
R8	R8D	R8W	R8B
R9	R9D	R9W	R9B
R10	R10D	R10W	R10B
R11	R11D	R11W	R11B
R12	R12D	R12W	R12B
R13	R13D	R13W	R13B
R14	R14D	R14W	R14B
R15	R15D	R15W	R15B

Instruction Pointer

RIP	EIP	IP	

Special Registers

- RFLAGS Register
  - A 64-bit wide register that contains various processor status flags and control bits
  - The low-order 32 bits of RFLAGS correspond to the EFLAGS register
  - The function of the auxiliary carry flag (AF), carry flag (CF), overflow flag (OF), parity flag (PF), sign flag (SF), and zero flag (ZF) are the same as EFLAGS
  - The high-order 32-bits of RFLAGS are reserved for future use
  - The pushfq and popfq instructions can be used to push or pop RFLAGS onto or from the stack
- Instruction Pointer Register
  - A 64-bit register contains the offset of the next instruction to be executed
  - RIP is implicitly manipulated by the control-transfer instructions call, ret, jmp, and jcc (like EIP)
  - The processor also uses the RIP register to support RIP-relative (or instruction pointer relative) addressing (an effective address is calculated using the contents of the RIP register and a signed 32-bit displacement value that's encoded within the instruction)
  - It is not possible for a task to directly access the RIP register

Assembly-to-C++ Statement Corollaries

Type	Example	Similar Statement
Immediate	movq $10, %rax	%rax = 10
	imulq $48, %rax	%rax *= 48
	shlq $2, %rcx	%rcx <<= 2
	xorl $8, %ecx	%ecx ^= 8
	subq $64, %rsp	%rsp -= 64
Register	movq %rsp, %rbp	%rbp = %rsp
	addq %r8, %rdi	%rdi += %r8
	imulq %rcx, %rax	%rax *= %rcx
	andq $1, %rdx	%rdx &= $1
Memory	movq -8(%rbp), %rsi	%rsi = *(%rbp + 8)
	addsd (%rsi,%rax), %xmm0	%xmm += *(%rsi + %rax)
	movsd %xmm0, 8(%rsi,%rax)	*(%rsi + %rax + 8) = %xmm0
	shlq $2, (%r8)	*(long *)%r8 <<= 2

AT&T vs. Intel Syntax

	AT&T	Intel
Parameter order	source, destination	destination, source
	addq $32, %rax	mov rax, 5
Parameter size	instruction suffix (q, l, w, b)	implied by register name (+ directives)
	addl $4, %rsp	add esp, 4
Sigils	$<immediate>, %<register>	auto-detected by assembler
Effective addresses	disp(base,index,scale)	[base+index*scale+disp]
	seg:disp(base,index,scale)	seg:[base+index*scale+disp]

Effective Addresses and Suffixes

- Syntax - disp(base,index,scale)
- All but base are optional.
- Disp specifies the displacement from the calculated effective address.
- The base register can be any general purpose register.
- The index register can be any general purpose register except %RSP.
- Scale is a factor by which index is to be multiplied before being added to base to specify the address of the operand.
- Valid factors include 1, 2, 4, and 8.
- The displacement is a constant 8-bit, 16-bit, or 32-bit signed offset that is encoded within the instruction.
- The final size of an effective address calculation is always 64 bits.

- Immediate data should not be prefixed with '$' when used for scale or disp.
- Scale and index default to 1 when not specified.

- Suffixes
  - q = quadword
  - l = long
  - w = word
  - b = byte
- Register can also dictate data type (e.g. FPU vs. GPR)

- NOTE: an optional segment register can be specified (e.g. seg:disp(base,index,scale)) when running in real or protected modes (see https://en.wikipedia.org/wiki/X86_memory_segmentation).

Common Instructions

- Data movement instructions:
  - mov - move
  - push - push on stack
  - pop - pop from stack
  - lea - load effective address
- Arithmetic and logic instructions:
  - add - integer addition
  - sub - integer subtraction
  - inc, dec - increment, decrement
  - imul - integer multiplication
  - idiv - integer division
  - and, or, xor - bitwise logical and, or, and exclusive or
  - not - bitwise logical not
  - neg - negate
  - shl, shr - shift left and right
- Control flow instructions:
  - jmp - jump
  - j<condition> - conditional jump
    - je - jump when equal
    - jne - jump when not equal
    - jz - jump when last result was zero
    - jg - jump when greater than
    - jge - jump when greater than or equal to
    - jl - jump when less than
    - jle - jump when less than or equal to
- cmp - compare
- call, ret - subroutine call and return

Some tips…

- When looking at memory, be aware of the direction the stack and heap grow and the endianness of your computer (little endian for my Intel)
- Classes/structs are laid out over increasing addresses while the stack grows down and the heap grows up (on my Mac)
- When in doubt, debug with hex literals to find the location in memory
- The Mac calculator's programmer mode comes in handy for calculating effective addresses
- Use https://godbolt.org! This helps with rapid iteration, checking the number of instructions for a particular construct, and seeing the relationship between expressions and the resulting assembly
- Some useful LLDB commands (don't forget that these only work with a debug build):
  - `im loo -rvs "regex"`
  - `frame variable`
  - `p &<var name>`
  - `mem r <addr>`

Common Declarations

```
TranslationUnitDecl
|-TypedefDecl implicit __int128_t '__int128'
| `-BuiltinType '__int128'

|-TypedefDecl implicit __uint128_t 'unsigned __int128'
| `-BuiltinType 'unsigned __int128'

|-TypedefDecl implicit __NSConstantString 'struct __NSConstantString_tag'

| `-RecordType 'struct __NSConstantString_tag'
|   `-CXXRecord '__NSConstantString_tag'

|-TypedefDecl implicit __builtin_ms_va_list 'char *'
| `-PointerType 'char *'
|   `-BuiltinType 'char'

|-TypedefDecl implicit __builtin_va_list 'struct __va_list_tag [1]'
| `-ConstantArrayType 'struct __va_list_tag [1]' 1
|   `-RecordType 'struct __va_list_tag'
|     `-CXXRecord '__va_list_tag'
```

- These are all types that are initialized in clang/lib/Sema/Sema.cpp Sema::Initialize() for various reasons and used in clang/include/clang/AST/ASTContext.h as well

Definition of struct X and Y

```
|-CXXRecordDecl referenced struct X definition
| |-CXXRecordDecl implicit struct X
| |-FieldDecl x 'double'
| |-FieldDecl y 'double'
| |-CXXConstructorDecl implicit used X 'void (void) noexcept' inline default trivial
| | `-CompoundStmt
| |-CXXConstructorDecl implicit constexpr X 'void (const struct X &)' inline default trivial noexcept-unevaluated
| | `-ParmVarDecl 'const struct X &'
| `-CXXConstructorDecl implicit constexpr X 'void (struct X &&)' inline default trivial noexcept-unevaluated
|   `-ParmVarDecl 'struct X &&’
```

```
|-CXXRecordDecl referenced struct Y definition
| |-CXXRecordDecl implicit referenced struct Y
| |-FieldDecl v ‘double'
| |-FieldDecl n 'struct Y *'
| |-CXXConstructorDecl implicit used Y 'void (void) noexcept' inline default trivial
| | `-CompoundStmt
| |-CXXConstructorDecl implicit constexpr Y 'void (const struct Y &)' inline default trivial noexcept-unevaluated
| | `-ParmVarDecl 'const struct Y &'
| `-CXXConstructorDecl implicit constexpr Y 'void (struct Y &&)' inline default trivial noexcept-unevaluated
|   `-ParmVarDecl 'struct Y &&'
```

Layout of struct X and Y

```
*** Dumping AST Record Layout
         0 | struct X
         0 |   double x
         8 |   double y
           | [sizeof=16, dsize=16, align=8,
           |  nvsize=16, nvalign=8]

…

Layout: <CGRecordLayout
  LLVMType:%struct.X = type { double, double }
  NonVirtualBaseLLVMType:%struct.X = type
{ double, double }
  IsZeroInitializable:1
  BitFields:[
]>
```

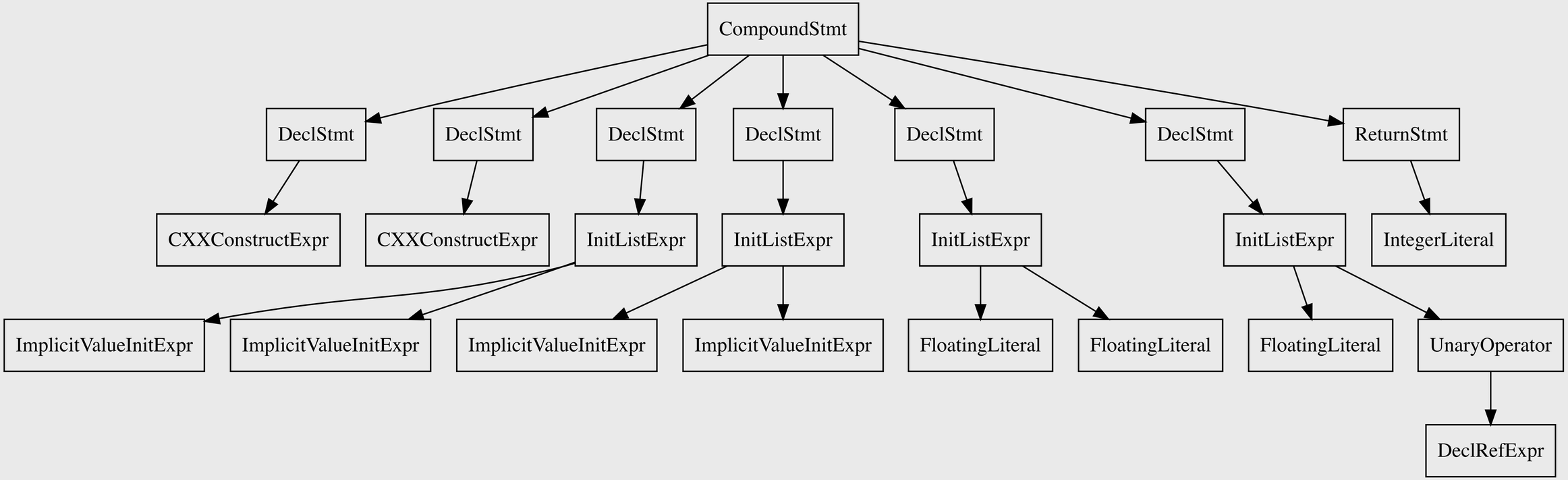
```
*** Dumping AST Record Layout
         0 | struct Y
         0 |   double v
         8 |   struct Y * n
           | [sizeof=16, dsize=16, align=8,
           |  nvsize=16, nvalign=8]

…

Layout: <CGRecordLayout
  LLVMType:%struct.Y = type { double, %struct.Y* }
  NonVirtualBaseLLVMType:%struct.Y = type
{ double, %struct.Y* }
  IsZeroInitializable:1
  BitFields:[
]>
```

int main()

```
`-FunctionDecl main 'int (void)'
  `-CompoundStmt
    |-DeclStmt
    | `-VarDecl x1 'struct X' callinit
    |   `-CXXConstructExpr 'struct X' 'void (void) noexcept'
    |-DeclStmt
    | `-VarDecl y1 'struct Y' callinit
    |   `-CXXConstructExpr 'struct Y' 'void (void) noexcept'
    |-DeclStmt
    | `-VarDecl x2 'struct X' listinit
    |   `-InitListExpr 'struct X'
    |     |-ImplicitValueInitExpr 'double'
    |     `-ImplicitValueInitExpr 'double'
    |-DeclStmt
    | `-VarDecl used y2 'struct Y' listinit
    |   `-InitListExpr 'struct Y'
    |     |-ImplicitValueInitExpr 'double'
    |     `-ImplicitValueInitExpr 'struct Y *'
    |-DeclStmt
    | `-VarDecl x3 'struct X' listinit
    |   `-InitListExpr 'struct X'
    |     |-FloatingLiteral 'double' 1.000000e+00
    |     `-FloatingLiteral 'double' 2.000000e+00
    |-DeclStmt
    | `-VarDecl y3 'struct Y' listinit
    |   `-InitListExpr 'struct Y'
    |     |-FloatingLiteral 'double' 3.000000e+00
    |     `-UnaryOperator 'struct Y *' prefix '&'
    |       `-DeclRefExpr 'struct Y' lvalue Var 0x7fa107002c28 'y2' 'struct Y'
    `-ReturnStmt
      `-IntegerLiteral 'int' 0
```

```
int main()
{
    X x1;

    Y y1;

    X x2{};

    Y y2{};

    X x3{1.0, 2.0};

    Y y3{3.0, &y2};

    return 0;
}
```

int main()


CompoundStmt
DeclStmt
DeclStmt
DeclStmt
DeclStmt
DeclStmt
DeclStmt
ReturnStmt
CXXConstructExpr
CXXConstructExpr
InitListExpr
InitListExpr
InitListExpr
InitListExpr
IntegerLiteral
ImplicitValueInitExpr
ImplicitValueInitExpr
ImplicitValueInitExpr
ImplicitValueInitExpr
FloatingLiteral
FloatingLiteral
FloatingLiteral
UnaryOperator
DeclRefExpr

More on the AST

- The three core classes to represent AST nodes are Decl, Stmt, and Type, which represent declarations, statements, and types, respectively
- Each C++ language construct is represented in clang by a C++ class which inherits from one of these core classes
- The top-level AST node is TranslationUnitDecl which is the root of all other AST nodes and represents an entire translation unit
- Can run clang in debug or use libclang and program against the AST API directly
  - see clang/Frontend/CompilerInstance.h and clang/Frontend/ASTConsumers.h for more info
  - http://clang.llvm.org/docs/IntroductionToTheClangAST.html

Common AST Classes

BinaryOperator - A builtin binary operation expression such as "x + y" or "x <= y”. This expression node kind describes a builtin binary operation, such as "x + y" for integer values "x" and "y". The operands will already have been converted to appropriate types (e.g., by performing promotions or conversions). In C++, where operators may be overloaded, a different kind of expression node (CXXOperatorCallExpr) is used to express the invocation of an overloaded operator with operator syntax. Within a C++ template, whether BinaryOperator or CXXOperatorCallExpr is used to store an expression "x + y" depends on the subexpressions for x and y. If neither x or y is type-dependent, and the "+" operator resolves to a built-in operation, BinaryOperator will be used to express the computation (x and y may still be value-dependent). If either x or y is type-dependent, or if the "+" resolves to an overloaded operator, CXXOperatorCallExpr will be used to express the computation. Definition at line 2967 of file Expr.h.

CXXMethodDecl - Represents a static or instance method of a struct/union/class. In the terminology of the C++ Standard, these are the (static and non-static) member functions, whether virtual or not. Definition at line 1833 of file DeclCXX.h.

CXXRecordDecl - Represents a C++ struct/union/class. Definition at line 267 of file DeclCXX.h.

CallExpr - Represents a function call (C99 6.5.2.2, C++ [expr.call]). CallExpr itself represents a normal function call, e.g., "f(x, 2)", while its subclasses may represent alternative syntax that (semantically) results in a function call. For example, CXXOperatorCallExpr is a subclass for overloaded operator calls that use operator syntax, e.g., "str1 + str2" to resolve to a function call. Definition at line 2206 of file Expr.h.

ClassTemplateSpecializationDecl - Represents a class template specialization, which refers to a class template with a given set of template arguments. Class template specializations represent both explicit specialization of class templates, as in the example below, and implicit instantiations of class templates.

template<typename T> class array;
template<>class array<bool> { };
// class template specialization
array<bool>
Definition at line 1630 of file DeclTemplate.h.

CompoundStmt - This represents a group of statements like { stmt stmt }. Definition at line 575 of file Stmt.h.

DeclRefExpr - A reference to a declared variable, function, enum, etc. [C99 6.5.1p2] This encodes all the information about how a declaration is referenced within an expression. There are several optional constructs attached to DeclRefExprs only when they apply in order to conserve memory. These are laid out past the end of the object, and flags in the DeclRefExprBitfield track whether they exist:

DeclRefExprBits.HasQualifier: Specifies when this declaration reference expression has a C++ nested-name-specifier.
DeclRefExprBits.HasFoundDecl: Specifies when this declaration reference expression has a record of a NamedDecl (different from the referenced ValueDecl) which was found during name lookup and/or overload resolution.
DeclRefExprBits.HasTemplateKWAndArgsInfo: Specifies when this declaration reference expression has an explicit C++ template keyword and/or template argument list.
DeclRefExprBits.RefersToEnclosingVariableOrCapture: Specifies when this declaration reference expression (validly) refers to an enclosed local or a captured variable.
Definition at line 953 of file Expr.h.

FunctionDecl - An instance of this class is created to represent a function declaration or definition. Since a given function can be declared several times in a program, there may be several FunctionDecls that correspond to that function. Only one of those FunctionDecls will be found when traversing the list of declarations in the context of the FunctionDecl (e.g., the translation unit); this FunctionDecl contains all of the information known about the function. Other, previous declarations of the function are available via the getPreviousDecl() chain. Definition at line 1618 of file Decl.h.

ImplicitCastExpr - Allows us to explicitly represent implicit type conversions, which have no direct representation in the original source code.

For example: converting T[]->T*, void f()->void (*f)(), float->double, short->int, etc.

In C, implicit casts always produce rvalues. However, in C++, an implicit cast whose result is being bound to a reference will be an lvalue or xvalue. For example:

class Base { };class Derived : public Base { };Derived &&ref();void f(Derived d) { Base& b = d; // initializer is an ImplicitCastExpr // to an lvalue of type Base Base&& r = ref(); // initializer is an ImplicitCastExpr // to an xvalue of type Base}

Definition at line 2804 of file Expr.h.

The AST Commands

- AST PDF - `clang -fsyntax-only -Xclang -ast-view --std=c++14 ${CXX_FLAGS} <in_file.cpp>`
  - NOTE: this command requires a debug build of clang/LLVM
- Text AST - `clang-check -p ${compile_commands_dir} -ast-dump <in_file.cpp> --extra-arg —std=c++14 --`
  - add `--extra-arg -fno-color-diagnostics` to remove colors
  - requires a compile_commands.json file for the related file
  - use set(CMAKE_EXPORT_COMPILE_COMMANDS ON) with CMake
- Class layouts - `clang -Xclang -fdump-record-layouts --std=c++14 ${CXX_FLAGS} <in_file.cpp>`
  - add `-fno-color-diagnostics` to remove colors

ABI Introduction

- https://libcxxabi.llvm.org/
  - Correctness as defined by the C++11 standard
  - Provide a portable sublayer to ease the porting of libc++
  - On Mac OS X, be ABI compatible with the existing low-level support
- https://libcxxabi.llvm.org/spec.html
- https://itanium-cxx-abi.github.io/cxx-abi/abi.html
  - The object code interfaces between user C++ code and the implementation-provided system and libraries
  - Includes memory layout for C++ data objects, including both predefined and user-defined data types, as well as internal compiler generated objects such as virtual tables
  - Includes function calling interfaces, exception handling interfaces, global naming, and various object code conventions

Name Mangling

- Entities with C linkage and global namespace variables are not mangled. Mangled names have the general structure:
  - <mangled-name> ::= _Z <encoding>
  - <encoding> ::= <function name> <bare-function-type>
    - ::= <data name>
    - ::= <special-name>
- A name is mangled by prefixing "_Z" to an encoding of its name, and in the case of functions its type (to support overloading).
- Constructors and destructors are simply special cases of <unqualified-name>, where the final <unqualified-name> of a nested name is replaced by one of the following:
  - <ctor-dtor-name>C1 - complete object constructor
  - <ctor-dtor-name>C2 - base object constructor
  - <ctor-dtor-name>C3 - complete object allocating constructor
  - <ctor-dtor-name>D0 - deleting destructor
  - <ctor-dtor-name>D1 - complete object destructor
  - <ctor-dtor-name>D2 - base object destructor
- https://itanium-cxx-abi.github.io/cxx-abi/abi.html#mangling
- https://itanium-cxx-abi.github.io/cxx-abi/abi-examples.html#mangling
- https://stackoverflow.com/questions/15452133/arm-c-abi-constructor-destructor-return-values
- https://stackoverflow.com/questions/6921295/dual-emission-of-constructor-symbols

f	C function or variable "f" or a global namespace variable "f"
_Z1fv	Ret? f(); or Ret? f(void);
_Z1fi	Ret? f(int);
_Z3foo3bar	Ret? foo(bar);
_ZN3FooIA4_iE3barE	Type? Foo<int[4]>::bar;
_ZTI7a_class	typeid(class a_class)

Calling Convention

- The OS X x86-64 function calling conventions are the same as the function calling conventions described in System V Application Binary Interface AMD64 Architecture Processor Supplement
  - https://developer.apple.com/library/content/documentation/DeveloperTools/Conceptual/LowLevelABI/130-IA-32_Function_Calling_Conventions/
  - https://software.intel.com/sites/default/files/article/402129/mpx-linux64-abi.pdf
- Calling conventions describe (from Wikipedia)
  - The order in which atomic (scalar) parameters, or individual parts of a complex parameter, are allocated
  - How parameters are passed (pushed on the stack, placed in registers, or a mix of both)
  - Which registers the callee must preserve for the caller
  - How the task of preparing the stack for, and restoring after, a function call is divided between the caller and the callee
- The prolog performs the following tasks:
  - Pushes the value of the stack frame pointer (EBP) onto the stack
  - Sets the stack frame pointer to the value of the stack pointer(ESP)
  - Pushes the values of the registers that must be preserved (EDI, ESI, and EBX) onto the stack
  - Allocates space in the stack frame for local storage
  - The epilog performs these tasks:
  - Deallocates the space used for local storage in the stack
  - Restores the preserved registers (EDI, ESI, EBX, EBP) by popping the values saved on the stack by the prolog
  - Returns
- See also
  - llvm/include/llvm/IR/CallingConv.h
  - llvm/lib/CodeGen/CallingConvLower.cpp

```
movq %rsi, -648(%rbp)         ## 8-byte Spill
movq %rdx, -656(%rbp)         ## 8-byte Spill
movq %rax, -664(%rbp)         ## 8-byte Spill
movq %rcx, -672(%rbp)
callq   __ZNKSt3__122__compressed_pair_elemIPiLi0ELb0EE5__getEv
```

```
   .cfi_startproc
## BB#0:                                ## %entry
   pushq   %rbp
Lcfi45:
   .cfi_def_cfa_offset 16
Lcfi46:
   .cfi_offset %rbp, -16
   movq %rsp, %rbp
Lcfi47:
   .cfi_def_cfa_register %rbp
      # pushq %rbx     ## save RBX
      # subq $24, %rsp   ## allocate space for local storage
```

```
.cfi_def_cfa_register %rbp
# addq $24, %rsp   ## deallocate space for local storage
movq %rdi, -8(%rbp)
movq -8(%rbp), %rax
popq %rbp
retq
.cfi_endproc
```

Types of Initialization

- Value initialization, e.g. std::string s{};
- Direct initialization, e.g. std::string s("hello");
- Copy initialization, e.g. std::string s = "hello";
- List initialization, e.g. std::string s{'a', 'b', 'c'};
- Aggregate initialization, e.g. char a[3] = {'a', 'b'};
- Reference initialization, e.g. char& c = a[0];
- If no initializer is provided, the rules of default initialization apply.
- Static initialization:
  - Constant initialization (generally at compile time)
  - Zero initialization takes place for all other non-local static and thread-local variables.
- http://en.cppreference.com/w/cpp/language/initialization
- See also section 11.6 - Initializers of n4659 (current draft standard).

Member Initialization in a Constructor

```
class Xyz {
private:
    int _v;

public:
    X x;
    Y y;
    Z z;

    Xyz() : _v{2048}, y{1024} {}

    ~Xyz() {}
};
```

```
|-CXXRecordDecl referenced class Xyz definition
| |-CXXRecordDecl implicit referenced class Xyz
| |-AccessSpecDecl private
| |-FieldDecl _v 'int'
| |-AccessSpecDecl public
| |-FieldDecl x 'struct X'
| |-FieldDecl y 'struct Y'
| |-FieldDecl z 'struct Z'
| |-CXXConstructorDecl used Xyz 'void (void)'
| | |-CXXCtorInitializer '_v' 'int'
| | | `-InitListExpr 'int'
| | |   `-IntegerLiteral 'int' 2048
| | |-CXXCtorInitializer Field 'x' 'struct X'
| | | `-CXXConstructExpr 'struct X' 'void (void)'
| | |-CXXCtorInitializer 'y' 'struct Y'
| | | `-CXXConstructExpr 'struct Y' 'void (int)'
| | |   `-IntegerLiteral 'int' 1024
| | |-CXXCtorInitializer Field 'z' 'struct Z'
| | | `-CXXConstructExpr 'struct Z' 'void (void)'
| | `-CompoundStmt
| |-CXXDestructorDecl used ~Xyz 'void (void) noexcept'
| | `-CompoundStmt
| `-CXXConstructorDecl implicit constexpr Xyz 'void (const class Xyz &)'
|                      inline default trivial noexcept-unevaluated
|   `-ParmVarDecl 'const class Xyz &'
```

Class with a Virtual Function

```
struct X {
    virtual void m1() const = 0;
};

void func(const X &x) {
    x.m1();
}

struct Y : public X {
    void m1() const override {}
};

struct Z : public X {
    void m1() const override {}
};

int main()
{
    Y y{};
    func(y);

    Z z{};
    func(z);
    return 0;
}
```

```
*** Dumping AST Record Layout
         0 | struct X
         0 |   (X vtable pointer)
           | [sizeof=8, dsize=8, align=8,
           |  nvsize=8, nvalign=8]

*** Dumping AST Record Layout
         0 | struct Y
         0 |   struct X (primary base)
         0 |     (X vtable pointer)
           | [sizeof=8, dsize=8, align=8,
           |  nvsize=8, nvalign=8]

*** Dumping AST Record Layout
         0 | struct Z
         0 |   struct X (primary base)
         0 |     (X vtable pointer)
           | [sizeof=8, dsize=8, align=8,
           |  nvsize=8, nvalign=8]
```

```
   .weak_def_can_be_hidden __ZN1ZC2Ev
   .p2align  4, 0x90
__ZN1ZC2Ev:                             ## @_ZN1ZC2Ev
   .cfi_startproc
## BB#0:                                ## %entry
   pushq %rbp
Lcfi21:
   .cfi_def_cfa_offset 16
Lcfi22:
   .cfi_offset %rbp, -16
   movq %rsp, %rbp
Lcfi23:
   .cfi_def_cfa_register %rbp
   subq $16, %rsp
   movq %rdi, -8(%rbp)
   movq -8(%rbp), %rdi
   movq %rdi, %rax
   movq %rdi, -16(%rbp)         ## 8-byte Spill
   movq %rax, %rdi
   callq__ZN1XC2Ev
   movq __ZTV1Z@GOTPCREL(%rip), %rax
   addq $16, %rax
   movq -16(%rbp), %rdi         ## 8-byte Reload
   movq %rax, (%rdi)
   addq $16, %rsp
   popq %rbp
   retq
   .cfi_endproc
```

```
__ZTV1Y:
   .quad   0
   .quad   __ZTI1Y
   .quad   __ZNK1Y2m1Ev

   .section  __TEXT,__const
   .globl  __ZTS1Y                  ## @_ZTS1Y
   .weak_definition  __ZTS1Y

__ZTV1X:
   .quad   0
   .quad   __ZTI1X
   .quad   ___cxa_pure_virtual

   .globl  __ZTV1Z                  ## @_ZTV1Z
   .weak_def_can_be_hidden  __ZTV1Z
   .p2align  3
__ZTV1Z:
   .quad   0
   .quad   __ZTI1Z
   .quad   __ZNK1Z2m1Ev

   .section  __TEXT,__const
   .globl  __ZTS1Z                  ## @_ZTS1Z
   .weak_definition  __ZTS1Z
```

```
function __ZNK1Y2m1Ev() { return rax; }
function __ZNK1Z2m1Ev() { return rax; }
```

```
function __ZN1XC2Ev() {
    *rdi = 0x100001088;
    return 0x100001088;
}
```

```
function __ZN1YC2Ev() {
    X::X();
    *rdi = 0x100001040;
    return 0x100001040;
}
```

```
function __ZN1ZC2Ev() {
    X::X();
    *rdi = 0x1000010a0;
    return 0x1000010a0;
}
```

```
void __cxa_pure_virtual()
{
    fprintf(stderr,
            "libc++abi.dylib: Pure virtual function called!\n");
    abort();
};
void Ym1(void *this) {}
void Zm1(void *this) {}
void (*__vtable_X[1])(void *) = {__cxa_pure_virtual};
void (*__vtable_Y[1])(void *) = {Ym1};
void (*__vtable_Z[1])(void *) = {Zm1};

X *XConstructor(X *this)
{
    *((method *)this) = (void *)__vtable_X;
    return this;
}

Y *YConstructor(Y *this)
{
    XConstructor((X *) this);
    *((method *)this) = (void *)__vtable_Y;
    return this;
}

Z *ZConstructor(Z *this)
{
    XConstructor((X *) this);
    *((method *)this) = (void *)__vtable_Z;
    return this;
}
```

Argument Initialization

```
struct X {
    int x = 0;

    X(int a) : x{a} {}

    X() = default;

    ~X() = default;
};

void func(X x)
{
    auto y = x.x;
}

int main()
{
    X x1 = 10;
    func(x1);
    func(10);

    X x2{};
    func(x2);
    func({});
    return 0;
}
```

```
void func(X x)
{
    int32_t y = *(int32_t *)(&x);
    return;
}


int main() {
    X x1;
    XConstructori(&x1, 0xA);
    func(x1);
    X __temp0;
    XConstructori(&__temp0, 0xA);
    func(__temp0);

    X x2;
    memset(&x2, 0x0, 0x4);
    XConstructor(&x2);
    func(x2);
    X __temp1;
    memset(&__temp1, 0x0, 0x4);
    XConstructor(&__temp1);
    func(__temp1);
    return 0x0;
}
```

```
|-FunctionDecl used func 'void (struct X)'
| |-ParmVarDecl used x 'struct X'
| `-CompoundStmt
|   `-DeclStmt
|     `-VarDecl y 'int':'int' cinit
|       `-ImplicitCastExpr 'int' <LValueToRValue>
|         `-MemberExpr 'int' lvalue .x 0x7f94cd065e28
|           `-DeclRefExpr 'struct X' lvalue ParmVar 0x7f94cd066438 'x' 'struct X'
`-FunctionDecl main 'int (void)'
  `-CompoundStmt
    |-DeclStmt
    | `-VarDecl used x1 'struct X' cinit
    |   `-ExprWithCleanups 'struct X'
    |     `-CXXConstructExpr 'struct X' 'void (const struct X &) noexcept' elidable
    |       `-MaterializeTemporaryExpr 'const struct X' lvalue
    |         `-ImplicitCastExpr 'const struct X' <NoOp>
    |           `-ImplicitCastExpr 'struct X' <ConstructorConversion>
    |             `-CXXConstructExpr 'struct X' 'void (int)'
    |               `-IntegerLiteral 'int' 10
    |-CallExpr 'void'
    | |-ImplicitCastExpr 'void (*)(struct X)' <FunctionToPointerDecay>
    | | `-DeclRefExpr 'void (struct X)' lvalue Function 'func' 'void (struct X)'
    | `-CXXConstructExpr 'struct X' 'void (const struct X &) noexcept'
    |   `-ImplicitCastExpr 'const struct X' lvalue <NoOp>
    |     `-DeclRefExpr 'struct X' lvalue Var 'x1' 'struct X'
    |-ExprWithCleanups 'void'
    | `-CallExpr 'void'
    |   |-ImplicitCastExpr 'void (*)(struct X)' <FunctionToPointerDecay>
    |   | `-DeclRefExpr 'void (struct X)' lvalue Function 0x7f94cd066500 'func' 'void (struct X)'
    |   `-CXXConstructExpr 'struct X' 'void (const struct X &) noexcept' elidable
    |     `-MaterializeTemporaryExpr 'const struct X' lvalue
    |       `-ImplicitCastExpr 'const struct X' <NoOp>
    |         `-ImplicitCastExpr 'struct X' <ConstructorConversion>
    |           `-CXXConstructExpr 'struct X' 'void (int)'
    |             `-IntegerLiteral 'int' 10
    |-DeclStmt
    | `-VarDecl used x2 'struct X' listinit
    |   `-CXXConstructExpr 'struct X' 'void (void) noexcept' zeroing
    |-CallExpr 'void'
    | |-ImplicitCastExpr 'void (*)(struct X)' <FunctionToPointerDecay>
    | | `-DeclRefExpr 'void (struct X)' lvalue Function 'func' 'void (struct X)'
    | `-CXXConstructExpr 'struct X' 'void (const struct X &) noexcept'
    |   `-ImplicitCastExpr 'const struct X' lvalue <NoOp>
    |     `-DeclRefExpr 'struct X' lvalue Var 'x2' 'struct X'
    |-CallExpr 'void'
    | |-ImplicitCastExpr 'void (*)(struct X)' <FunctionToPointerDecay>
    | | `-DeclRefExpr 'void (struct X)' lvalue Function 0x7f94cd066500 'func' 'void (struct X)'
    | `-CXXConstructExpr 'struct X' 'void (void) noexcept' zeroing
    `-ReturnStmt
      `-IntegerLiteral 'int' 0
```

```
subq $64, %rsp
leaq -8(%rbp), %rdi
movl $10, %esi
movl $0, -4(%rbp)
callq   __ZN1XC1Ei
movl -8(%rbp), %esi
movl %esi, -16(%rbp)
movl -16(%rbp), %edi
callq   __Z4func1X
leaq -24(%rbp), %rdi
movl $10, %esi
callq   __ZN1XC1Ei
movl -24(%rbp), %edi
callq   __Z4func1X
leaq -32(%rbp), %rax
xorl %esi, %esi
movl $4, %edi
movl %edi, %edx
movq %rax, %rcx
movq %rcx, %rdi
movq %rax, -56(%rbp)
callq   _memset
movq -56(%rbp), %rdi
callq   __ZN1XC1Ev
movl -32(%rbp), %esi
movl %esi, -40(%rbp)
movl -40(%rbp), %edi
callq   __Z4func1X
leaq -48(%rbp), %rax
xorl %esi, %esi
movl $4, %edi
movl %edi, %edx
movq %rax, %rcx
movq %rcx, %rdi
movq %rax, -64(%rbp)
callq   Mmes.
movq -64(%rbp), %rdi
callq   __ZN1XC1Ev
movl -48(%rbp), %edi
callq   __Z4func1X
xorl %eax, %eax
addq $64, %rsp
popq %rbp
retq
.cfi_endproc
```

portions adapted from https://jonasdevlieghere.com/order-your-members/

Data Member Layout

```cpp
// example 1
struct S1 {
    char c;
    int i;
    short s;
};

struct S2 {
    int i;
    short s;
    char c;
};

struct S3 {
    int i;
    short s;
    char c;
};

// example 2
struct X {
    std::int64_t a[13];
    std::int32_t b;
};

struct Y {
    X x;
    std::int64_t a;
    std::int64_t b;
    bool f;
};

struct Z : X {
    bool f;
    std::int64_t a;
    std::int64_t b;
};

// example 3
struct alignas(64) S64 {};

// example 4
class A {};
class B : public virtual A {};
class C : public virtual A {};
class D : public B, public C {};

// example 5
class P1 {
private:
    double _x;
    static std::vector<P1 *> &_free_list;
    double _y;
    static const int _chunk_size = 250;
    double _z;
};


class P2 {
private:
    double _x;
    static std::vector<P2 *> &_free_list;
private:
    double _y;
    static const int _chunk_size = 250;
private:
    double _z;
};

int main() {
    // example 1
    const auto s1 = sizeof(S1);
    const auto s2 = sizeof(S2);
    const auto s3 = sizeof(S3);

    // example 2
    const auto x = sizeof(X);
    const auto y = sizeof(Y);
    const auto z = sizeof(Z);

    // example 3
    const auto s64 = sizeof(S64);

    // example 4
    const auto a = sizeof(A);
    const auto b = sizeof(B);
    const auto c = sizeof(C);
    const auto d = sizeof(D);

    // example 5
    const auto p1 = sizeof(P1);
    const auto p2 = sizeof(P2);
    return 0;
}
```

```
*** Dumping AST Record Layout
         0 | struct S1
         0 |   char c
         4 |   int i
         8 |   short s
           | [sizeof=12, dsize=12, align=4,
           |  nvsize=12, nvalign=4]

*** Dumping AST Record Layout
         0 | struct S2
         0 |   int i
         4 |   short s
         6 |   char c
           | [sizeof=8, dsize=8, align=4,
           |  nvsize=8, nvalign=4]

*** Dumping AST Record Layout
         0 | struct S3
         0 |   int i
         4 |   short s
         6 |   char c
           | [sizeof=8, dsize=8, align=4,
           |  nvsize=8, nvalign=4]

*** Dumping AST Record Layout
         0 | struct X
         0 |   std::int64_t [13] a
       104 |   std::int32_t b
           | [sizeof=112, dsize=112, align=8,
           |  nvsize=112, nvalign=8]

*** Dumping AST Record Layout
         0 | struct Y
         0 |   struct X x
         0 |     std::int64_t [13] a
       104 |     std::int32_t b
       112 |   std::int64_t a
       120 |   std::int64_t b
       128 |   _Bool f
           | [sizeof=136, dsize=136, align=8,
           |  nvsize=136, nvalign=8]

*** Dumping AST Record Layout
         0 | struct Z
         0 |   struct X (base)
         0 |     std::int64_t [13] a
       104 |     std::int32_t b
       112 |   _Bool f
       120 |   std::int64_t a
       128 |   std::int64_t b
           | [sizeof=136, dsize=136, align=8,
           |  nvsize=136, nvalign=8]

*** Dumping AST Record Layout
         0 | struct S64 (empty)
           | [sizeof=64, dsize=64, align=64,
           |  nvsize=64, nvalign=64]

*** Dumping AST Record Layout
         0 | class A (empty)
           | [sizeof=1, dsize=1, align=1,
           |  nvsize=1, nvalign=1]

*** Dumping AST Record Layout
         0 | class B
         0 |   (B vtable pointer)
         0 |   class A (virtual base) (empty)
           | [sizeof=8, dsize=8, align=8,
           |  nvsize=8, nvalign=8]

*** Dumping AST Record Layout
         0 | class C
         0 |   (C vtable pointer)
         0 |   class A (virtual base) (empty)
           | [sizeof=8, dsize=8, align=8,
           |  nvsize=8, nvalign=8]

*** Dumping AST Record Layout
         0 | class D
         0 |   class B (primary base)
         0 |     (B vtable pointer)
         8 |   class C (base)
         8 |     (C vtable pointer)
         0 |   class A (virtual base) (empty)
           | [sizeof=16, dsize=16, align=8,
           |  nvsize=16, nvalign=8]

*** Dumping AST Record Layout
         0 | class P1
         0 |   double _x
         8 |   double _y
        16 |   double _z
           | [sizeof=24, dsize=24, align=8,
           |  nvsize=24, nvalign=8]

*** Dumping AST Record Layout
         0 | class P2
         0 |   double _x
         8 |   double _y
        16 |   double _z
           | [sizeof=24, dsize=24, align=8,
           |  nvsize=24, nvalign=8]
```

```
  .globl  _main
  .p2align  4, 0x90
_main:
  .cfi_startproc
  pushq   %rbp
Lcfi0:
  .cfi_def_cfa_offset 16
Lcfi1:
  .cfi_offset %rbp, -16
  movq %rsp, %rbp
Lcfi2:
  .cfi_def_cfa_register %rbp
  xorl %eax, %eax
  movl $0, -4(%rbp)
  movq $12, -16(%rbp)
  movq $8, -24(%rbp)
  movq $8, -32(%rbp)
  movq $112, -40(%rbp)
  movq $136, -48(%rbp)
  movq $136, -56(%rbp)
  movq $64, -64(%rbp)
  movq $1, -72(%rbp)
  movq $8, -80(%rbp)
  movq $8, -88(%rbp)
  movq $16, -96(%rbp)
  movq $24, -104(%rbp)
  movq $24, -112(%rbp)
  popq %rbp
  retq
  .cfi_endproc
```

sizeof, dsize, align, nvsize, nvalign

- Information specified about the data layout of an object (O) in the ABI:
- sizeof(O) - the size of an object.
- align(O) - the alignment of an object.
- offset(C) - the offset within O of each data component C, i.e. base or member.
- Additional information specified for purposes internal to the specification:
  - dsize(O) - the data size of an object, which is the size of O without tail padding.
- nvsize(O) - the non-virtual size of an object, which is the size of O without virtual bases.
- nvalign(O) - the non-virtual alignment of an object, which is the alignment of O without virtual bases.

Adding Polymorphism

```
// example 1
class P2 {
public:
    P2(double x, double y) : _x{x}, _y{y} {};

    // getters/setters _x and _y

    virtual double z() const
    {
        return 0.0;
    };

    virtual void z(double) {};

    virtual void
    operator+=(const P2 &rhs)
    {
        _x += rhs._x;
        _y += rhs._y;
    }

protected:
    double _x = 0.0;
    double _y = 0.0;
};



class P3 : public P2 {
public:
    P3(double x, double y, double z) : P2{x, y},
_z{z} {};

    double z() const
    {
        return _z;
    }

    void z(double z)
    {
        _z = z;
    }

    void operator+=(const P2 &rhs)
    {
        P2::operator+=(rhs);
        _z += rhs.z();
    }

protected:
    double _z;
};

void f(P2 &p1, P2 &p2)
{
    p1 += p2;
}

// example 2
struct C {
    int d1;
    int d2;
};

class V : public C {
public:
    virtual void m() {}
    // ...
private:
    int d3;
};

int main()
{
    // example 1
    P3 p3_1{1.0, 1.0, 1.0};
    P3 p3_2{2.0, 2.0, 2.0};
    f(p3_1, p3_2);

    // example 2
    C *p = new V;
    return 0;
}
```

```
typedef struct _P2 P2;

// added for syntactic convenience
typedef void (*pmethod)(P2 *, P2 *);

typedef struct _P2 {
    pmethod *__vptr;
    double _x;
    double _y;
} P2;

void P2operator_plus_equal(P2 *this, P2 *arg0)
{
    double __temp0 = *((double *) (this + 0x8));
    double __temp1 = *((double *) (arg0 + 0x8));
    *((double *) (this + 0x8)) = __temp0 + __temp1;
    __temp0 = *((double *) (this + 0x10));
    __temp1 = *((double *) (arg0 + 0x10));
    *((double *) (this + 0x10)) = __temp0 + __temp1;
}

void (*__vtable_P2[1])(P2 *, P2 *) = {P2operator_plus_equal};

P2 *P2Constructordd(P2 *this, double arg0, double arg1)
{
    *((pmethod *) this) = (void *) __vtable_P2;
    *((double *) ((void *) this + 0x8)) = arg0;
    *((double *) ((void *) this + 0x10)) = arg1;
    return this;
}

typedef struct _P3 {
    P2 primary_base;
    double _z;
} P3;

void P3operator_plus_equal(P2 *this, P2 *arg0)
{
    P2operator_plus_equal(this, arg0);
    double __temp0 = *((double *) (this + 0x18));
    double __temp1 = *((double *) (arg0 + 0x18));
    *((double *) (this + 0x18)) = __temp0 + __temp1;
}

void (*__vtable_P3[1])(P2 *, P2 *) = {P3operator_plus_equal};

P3 *P3Constructorddd(P3 *this, double arg0, double arg1, double arg2)
{
    P2Constructordd((P2 *) this, arg0, arg1);
    *((pmethod *) this) = (void *) __vtable_P3;
    *((double *) ((void *) this + 0x18)) = arg2;
    return this;
}

void f(P2 *arg0, P2 *arg1)
{
    (*(*(pmethod **) arg0))(arg0, arg1);
}

// added for syntactic convenience
typedef void (*vmethod)(void *);

typedef struct _C {
    int d1;
    int d2;
} C;

typedef struct _V {
    vmethod *__vptr;
    C base;
    int d3;
} V;

void Vm(void *this) {}

void (*__vtable_V[1])(void *) = {Vm};

V *VConstructor(V *this)
{
    *((vmethod *) this) = (void *) __vtable_V;
    return this;
}

int main()
{
    P3 p3_1;
    P3Constructorddd(&p3_1, 1.0, 1.0, 1.0);
    P3 p3_2;
    P3Constructorddd(&p3_2, 2.0, 2.0, 2.0);
    f((P2 *) &p3_1, (P2 *) &p3_2);

    C *p;
    void *__temp0 = operator_new(0x18);
    void *__temp1 = __temp0;
    __temp0 = VConstructor(__temp0);
    void *__temp2 = 0x0;
    if (__temp0 != 0x0) {
        __temp2 = __temp1 + 0x8;
    }
    p = __temp2;
    __temp0 = 0x0;
    return 0x0;
}
```

Multiple Inheritance

```cpp
struct P1 {
    double x;
};

struct P2 {
    double y;
};

struct P3 : public P1, public P2 {
    double z;
};

double func1(const P1 &p)
{
    return p.x * p.x;
}

double func2(const P2 &p)
{
    return p.y * p.y;
}

double func3(const P3 &p)
{
    return std::sqrt(func1(p) + func2(p));
}

int main()
{
    P3 p3{};
    P2 *pv = &p3;

    auto v = func3(p3);
    return 0;
}
```

```
*** Dumping AST Record Layout
         0 | struct P1
         0 |   double x
           | [sizeof=8, dsize=8, align=8,
           |  nvsize=8, nvalign=8]

*** Dumping AST Record Layout
         0 | struct P2
         0 |   double y
           | [sizeof=8, dsize=8, align=8,
           |  nvsize=8, nvalign=8]

*** Dumping AST Record Layout
         0 | struct P3
         0 |   struct P1 (base)
         0 |     double x
         8 |   struct P2 (base)
         8 |     double y
        16 |   double z
           | [sizeof=24, dsize=24, align=8,
           |  nvsize=24, nvalign=8]
```

```
|-FunctionDecl used func3 'double (const struct P3 &)'
| |-ParmVarDecl used p 'const struct P3 &'
| `-CompoundStmt
|   `-ReturnStmt
|     `-CallExpr 'double'
|       |-ImplicitCastExpr 'double (*)(double)' <FunctionToPointerDecay>
|       | `-DeclRefExpr 'double (double)' lvalue Function 0x7fa3330ec1c0 'sqrt' 'double (double)' (UsingShadow 'sqrt')
|       `-BinaryOperator 'double' '+'
|         |-CallExpr 'double'
|         | |-ImplicitCastExpr 'double (*)(const struct P1 &)' <FunctionToPointerDecay>
|         | | `-DeclRefExpr 'double (const struct P1 &)' lvalue Function 'func1' 'double (const struct P1 &)'
|         | `-ImplicitCastExpr 'const struct P1' lvalue <DerivedToBase (P1)>
|         |   `-DeclRefExpr 'const struct P3' lvalue ParmVar 0x7fa333932af8 'p' 'const struct P3 &'
|         `-CallExpr 'double'
|           |-ImplicitCastExpr 'double (*)(const struct P2 &)' <FunctionToPointerDecay>
|           | `-DeclRefExpr 'double (const struct P2 &)' lvalue Function 'func2' 'double (const struct P2 &)'
|           `-ImplicitCastExpr 'const struct P2' lvalue <DerivedToBase (P2)>
|             `-DeclRefExpr 'const struct P3' lvalue ParmVar 0x7fa333932af8 'p' 'const struct P3 &'
`-FunctionDecl main 'int (void)'
  `-CompoundStmt
    |-DeclStmt
    | `-VarDecl used p3 'struct P3' listinit
    |   `-CXXConstructExpr 'struct P3' 'void (void) noexcept' zeroing
    |-DeclStmt
    | `-VarDecl pv 'struct P2 *' cinit
    |   `-ImplicitCastExpr 'struct P2 *' <DerivedToBase (P2)>
    |     `-UnaryOperator 'struct P3 *' prefix '&'
    |       `-DeclRefExpr 'struct P3' lvalue Var 0x7fa3339333a8 'p3' 'struct P3'
    |-DeclStmt
    | `-VarDecl v 'double':'double' cinit
    |   `-CallExpr 'double'
    |     |-ImplicitCastExpr 'double (*)(const struct P3 &)' <FunctionToPointerDecay>
    |     | `-DeclRefExpr 'double (const struct P3 &)' lvalue Function 'func3' 'double (const struct P3 &)'
    |     `-ImplicitCastExpr 'const struct P3' lvalue <NoOp>
    |       `-DeclRefExpr 'struct P3' lvalue Var 'p3' 'struct P3'
    `-ReturnStmt
      `-IntegerLiteral 'int' 0
```

Non-Static Member Functions

```
double P3::m1() const
{
    return std::sqrt(this->_x * this->_x +
            this->_y * this->_y + this->_z * this->_z);
}

double nm1(const P3 *self)
{
    return std::sqrt(self->_x * self->_x +
            self->_y * self->_y + self->_z * self->_z);
}

int main()
{
    P3 p3{1.0, 1.0, 1.0};
    p3.m1();
    nm1(&p3);
    auto pp3 = &p3;
    nm1(pp3);
    pp3->m1();

    return 0;
}
```

```
double P3m1(P3 *this)
{
    double __temp0 = this->_x * this->_x;
    double __temp1 = this->_y * this->_y;
    __temp0 = __temp0 + __temp1;
    __temp0 = __temp0 + this->_z * this->_z;
    return sqrt(__temp0);
}

double nm1(const P3 *self)
{
    double __temp0 = self->_x * self->_x;
    double __temp1 = self->_y * self->_y;
    __temp0 = __temp0 + __temp1;
    __temp0 = __temp0 + self->_z * self->_z;
    return sqrt(__temp0);
}
```

```
|-CXXMethodDecl parent 0x7fba47c35a88 prev 0x7fba47c361a8 2 used m1 'double (void) const'
| `-CompoundStmt
|   `-ReturnStmt
|     `-CallExpr 'double'
|       |-ImplicitCastExpr 'double (*)(double)' <FunctionToPointerDecay>
|       | `-DeclRefExpr 'double (double)' lvalue Function 'sqrt' 'double (double)' (UsingShadow 'sqrt')
|       `-BinaryOperator 'double' '+'
|         |-BinaryOperator 'double' '+'
|         | |-BinaryOperator 'double' '*'
|         | | |-ImplicitCastExpr 'double' <LValueToRValue>
|         | | | `-MemberExpr 'const double' lvalue ->_x 0x7fba47c36280
|         | | |   `-CXXThisExpr 'const class P3 *' this
|         | | `-ImplicitCastExpr 'double' <LValueToRValue>
|         | |   `-MemberExpr 'const double' lvalue ->_x 0x7fba47c36280
|         | |     `-CXXThisExpr 'const class P3 *' this
|         | `-BinaryOperator 'double' '*'
|         |   |-ImplicitCastExpr 'double' <LValueToRValue>
|         |   | `-MemberExpr 'const double' lvalue ->_y 0x7fba47c362e0
|         |   |   `-CXXThisExpr 'const class P3 *' this
|         |   `-ImplicitCastExpr 'double' <LValueToRValue>
|         |     `-MemberExpr 'const double' lvalue ->_y 0x7fba47c362e0
|         |       `-CXXThisExpr 'const class P3 *' this
|         `-BinaryOperator 'double' '*'
|           |-ImplicitCastExpr 'double' <LValueToRValue>
|           | `-MemberExpr 'const double' lvalue ->_z 0x7fba47c36340
|           |   `-CXXThisExpr 'const class P3 *' this
|           `-ImplicitCastExpr 'double' <LValueToRValue>
|             `-MemberExpr 'const double' lvalue ->_z 0x7fba47c36340
|               `-CXXThisExpr 'const class P3 *' this
|-FunctionDecl prev 0x7fba47c36468 used nm1 'double (const class P3 *)'
| |-ParmVarDecl used self 'const class P3 *'
| `-CompoundStmt
|   `-ReturnStmt
|     `-CallExpr 'double'
|       |-ImplicitCastExpr 'double (*)(double)' <FunctionToPointerDecay>
|       | `-DeclRefExpr 'double (double)' lvalue Function 'sqrt' 'double (double)' (UsingShadow 'sqrt')
|       `-BinaryOperator 'double' '+'
|         |-BinaryOperator 'double' '+'
|         | |-BinaryOperator 'double' '*'
|         | | |-ImplicitCastExpr 'double' <LValueToRValue>
|         | | | `-MemberExpr 'const double' lvalue ->_x 0x7fba47c36280
|         | | |   `-ImplicitCastExpr 'const class P3 *' <LValueToRValue>
|         | | |     `-DeclRefExpr 'const class P3 *' lvalue ParmVar 0x7fba47c370e8 'self' 'const class P3 *'
|         | | `-ImplicitCastExpr 'double' <LValueToRValue>
|         | |   `-MemberExpr 'const double' lvalue ->_x 0x7fba47c36280
|         | |     `-ImplicitCastExpr 'const class P3 *' <LValueToRValue>
|         | |       `-DeclRefExpr 'const class P3 *' lvalue ParmVar 0x7fba47c370e8 'self' 'const class P3 *'
|         | `-BinaryOperator 'double' '*'
|         |   |-ImplicitCastExpr 'double' <LValueToRValue>
|         |   | `-MemberExpr 'const double' lvalue ->_y 0x7fba47c362e0
|         |   |   `-ImplicitCastExpr 'const class P3 *' <LValueToRValue>
|         |   |     `-DeclRefExpr 'const class P3 *' lvalue ParmVar 0x7fba47c370e8 'self' 'const class P3 *'
|         |   `-ImplicitCastExpr 'double' <LValueToRValue>
|         |     `-MemberExpr 'const double' lvalue ->_y 0x7fba47c362e0
|         |       `-ImplicitCastExpr 'const class P3 *' <LValueToRValue>
|         |         `-DeclRefExpr 'const class P3 *' lvalue ParmVar 0x7fba47c370e8 'self' 'const class P3 *'
|         `-BinaryOperator 'double' '*'
|           |-ImplicitCastExpr 'double' <LValueToRValue>
|           | `-MemberExpr 'const double' lvalue ->_z 0x7fba47c36340
|           |   `-ImplicitCastExpr 'const class P3 *' <LValueToRValue>
|           |     `-DeclRefExpr 'const class P3 *' lvalue ParmVar 0x7fba47c370e8 'self' 'const class P3 *'
|           `-ImplicitCastExpr 'double' <LValueToRValue>
|             `-MemberExpr 'const double' lvalue ->_z 0x7fba47c36340
|               `-ImplicitCastExpr 'const class P3 *' <LValueToRValue>
|                 `-DeclRefExpr 'const class P3 *' lvalue ParmVar 0x7fba47c370e8 'self' 'const class P3 *'
```

Static Member Functions

```
class P3 {
public:
    P3() : _x{0.0}, _y{0.0}, _z{0.0}
    {
        ++_object_count;
    }

    static unsigned long object_count();

private:
    double _x;
    double _y;
    double _z;
    static unsigned long _object_count;
};

unsigned long P3::_object_count = 0;

unsigned long P3::object_count()
{
    return _object_count;
}

int main()
{
    P3 p3_1{};
    P3 p3_2{};
    P3 p3_3{};

    const auto oc = P3::object_count();
    return 0;
}
```

```
// accessed as an address to global storage
unsigned long P3_object_count = 0;

unsigned long P3object_count()
{
    return P3_object_count;
}

P3 *P3Constructor(P3 *this)
{
    *(double *) this = 0.;
    *(double *) ((void *) this + 0x8) = 0.;
    *(double *) ((void *) this + 0x10) = 0.;
    P3_object_count = P3_object_count + 0x1;
    return this;
}
```

```
|-CXXRecordDecl referenced class P3 definition
| |-CXXRecordDecl implicit referenced class P3
| |-AccessSpecDecl public
| |-CXXConstructorDecl used P3 'void (void)'
| | |-CXXCtorInitializer Field 0x7fb87504d6c0 '_x' 'double'
| | | `-InitListExpr 'double'
| | |   `-FloatingLiteral 'double' 0.000000e+00
| | |-CXXCtorInitializer Field 0x7fb87504d720 '_y' 'double'
| | | `-InitListExpr 'double'
| | |   `-FloatingLiteral 'double' 0.000000e+00
| | |-CXXCtorInitializer Field 0x7fb87504d780 '_z' 'double'
| | | `-InitListExpr 'double'
| | |   `-FloatingLiteral 'double' 0.000000e+00
| | `-CompoundStmt
| |   `-UnaryOperator 'unsigned long' lvalue prefix '++'
| |     `-DeclRefExpr 'unsigned long' lvalue Var 0x7fb87504d7e0 '_object_count' 'unsigned long'
| |-CXXMethodDecl used object_count 'unsigned long (void)' static
| |-AccessSpecDecl private
| |-FieldDecl _x 'double'
| |-FieldDecl _y 'double'
| |-FieldDecl _z 'double'
| |-VarDecl used _object_count 'unsigned long' static
| |-CXXConstructorDecl implicit constexpr P3 'void (const class P3 &)' inline default trivial noexcept-unevaluated
| | `-ParmVarDecl 'const class P3 &'
| `-CXXConstructorDecl implicit constexpr P3 'void (class P3 &&)' inline default trivial noexcept-unevaluated
|   `-ParmVarDecl 'class P3 &&'
|-VarDecl parent 0x7fb87504d258 prev 0x7fb87504d7e0 used _object_count 'unsigned long' cinit
| `-ImplicitCastExpr 'unsigned long' <IntegralCast>
|   `-IntegerLiteral 'int' 0
|-CXXMethodDecl parent 0x7fb87504d258 prev 0x7fb87504d5e8 used object_count 'unsigned long (void)'
| `-CompoundStmt
|   `-ReturnStmt
|     `-ImplicitCastExpr 'unsigned long' <LValueToRValue>
|       `-DeclRefExpr 'unsigned long' lvalue Var 0x7fb87504dba0 '_object_count' 'unsigned long'
`-FunctionDecl main 'int (void)'
  `-CompoundStmt
    |-DeclStmt
    | `-VarDecl p3_1 'class P3' listinit
    |   `-CXXConstructExpr 'class P3' 'void (void)'
    |-DeclStmt
    | `-VarDecl p3_2 'class P3' listinit
    |   `-CXXConstructExpr 'class P3' 'void (void)'
    |-DeclStmt
    | `-VarDecl p3_3 'class P3' listinit
    |   `-CXXConstructExpr 'class P3' 'void (void)'
    |-DeclStmt
    | `-VarDecl oc 'const unsigned long':'const unsigned long' cinit
    |   `-CallExpr 'unsigned long'
    |     `-ImplicitCastExpr 'unsigned long (*)(void)' <FunctionToPointerDecay>
    |       `-DeclRefExpr 'unsigned long (void)' lvalue CXXMethod 0x7fb87504dcb0 'object_count' 'unsigned long (void)'
    `-ReturnStmt
      `-IntegerLiteral 'int' 0
```

Virtual Functions Under Virtual Inheritance

```cpp
struct B {
    int n;

    virtual void m() {
        n = 1;
    }
};

class X : public virtual B {
    virtual void m() override {
        B::n = 2;
    }
};

class Y : virtual public B {
    virtual void m() override {
        B::n = 3;
    }
};

class Z : public B {
    virtual void m() override {
        B::n = 4;
    }
};

struct AA : X, Y, Z {
    void m() override
    {
        X::n = 5; // modifies the virtual B subobject's member
        Y::n = 6; // modifies the same virtual B subobject's member
        Z::n = 7; // modifies the non-virtual B subobject's member
    }
};

int main()
{
    AA aa{};
    aa.m();
    return 0;
}
```

```
|-CXXRecordDecl referenced struct AA definition
| |-public 'class X'
| |-public 'class Y'
| |-public 'class Z'
| |-CXXRecordDecl implicit struct AA
| |-CXXMethodDecl used m 'void (void)'
| | |-CompoundStmt
| | | |-BinaryOperator 'int' lvalue '='
| | | | |-MemberExpr 'int' lvalue ->n 0x7f835904be28
| | | | | `-ImplicitCastExpr 'struct B *' <UncheckedDerivedToBase (virtual B)>
| | | | |   `-ImplicitCastExpr 'class X *' <UncheckedDerivedToBase (X)>
| | | | |     `-CXXThisExpr 'struct AA *' this
| | | | `-IntegerLiteral 'int' 5
| | | |-BinaryOperator 'int' lvalue '='
| | | | |-MemberExpr 'int' lvalue ->n 0x7f835904be28
| | | | | `-ImplicitCastExpr 'struct B *' <UncheckedDerivedToBase (virtual B)>
| | | | |   `-ImplicitCastExpr 'class Y *' <UncheckedDerivedToBase (Y)>
| | | | |     `-CXXThisExpr 'struct AA *' this
| | | | `-IntegerLiteral 'int' 6
| | | `-BinaryOperator 'int' lvalue '='
| | |   |-MemberExpr 'int' lvalue ->n 0x7f835904be28
| | |   | `-ImplicitCastExpr 'struct B *' <UncheckedDerivedToBase (B)>
| | |   |   `-ImplicitCastExpr 'class Z *' <UncheckedDerivedToBase (Z)>
| | |   |     `-CXXThisExpr 'struct AA *' this
| | |   `-IntegerLiteral 'int' 7
| | `-OverrideAttr
| |-CXXMethodDecl > implicit operator= 'struct AA &(const struct AA &)' inline default noexcept-unevaluated
0x7f835981a868
| | `-ParmVarDecl 'const struct AA &'
| |-CXXMethodDecl > implicit operator= 'struct AA &(struct AA &&)' inline default noexcept-unevaluated
| | `-ParmVarDecl 'struct AA &&'
| |-CXXDestructorDecl implicit ~AA 'void (void)' inline default trivial noexcept-unevaluated
| |-CXXConstructorDecl implicit used AA 'void (void) noexcept' inline default
| | |-CXXCtorInitializer 'struct B'
| | | `-CXXConstructExpr 'struct B' 'void (void) noexcept'
| | |-CXXCtorInitializer 'class X'
| | | `-CXXConstructExpr 'class X' 'void (void) noexcept'
| | |-CXXCtorInitializer 'class Y'
| | | `-CXXConstructExpr 'class Y' 'void (void) noexcept'
| | |-CXXCtorInitializer 'class Z'
| | | `-CXXConstructExpr 'class Z' 'void (void) noexcept'
| | `-CompoundStmt
```

```
*** Dumping AST Record Layout
         0 | struct B
         0 |   (B vtable pointer)
         8 |   int n
           | [sizeof=16, dsize=12, align=8,
           |  nvsize=12, nvalign=8]

*** Dumping AST Record Layout
         0 | class X
         0 |   (X vtable pointer)
         8 |   struct B (virtual base)
         8 |     (B vtable pointer)
        16 |     int n
           | [sizeof=24, dsize=20, align=8,
           |  nvsize=8, nvalign=8]

*** Dumping AST Record Layout
         0 | class Y
         0 |   (Y vtable pointer)
         8 |   struct B (virtual base)
         8 |     (B vtable pointer)
        16 |     int n
           | [sizeof=24, dsize=20, align=8,
           |  nvsize=8, nvalign=8]

*** Dumping AST Record Layout
         0 | class Z
         0 |   struct B (primary base)
         0 |     (B vtable pointer)
         8 |     int n
           | [sizeof=16, dsize=12, align=8,
           |  nvsize=12, nvalign=8]

*** Dumping AST Record Layout
         0 | struct AA
         0 |   class X (primary base)
         0 |     (X vtable pointer)
         8 |   class Y (base)
         8 |     (Y vtable pointer)
        16 |   class Z (base)
        16 |     struct B (primary base)
        16 |       (B vtable pointer)
        24 |       int n
        32 |   struct B (virtual base)
        32 |     (B vtable pointer)
        40 |     int n
           | [sizeof=48, dsize=44, align=8,
           |  nvsize=28, nvalign=8]
```

Inline Functions

```cpp
// NOTE: getters and setters will not be inlined until the -O2
// optimization flag
class P {
public:
    P(double x, double y) : _x{x}, _y{y} {}

    P() = default;

    ~P() = default;

    // getters/setters for _x & _y

private:
    double _x;
    double _y;

    friend
    P operator+(const P & lhs, const P & rhs);
};

P operator+(const P &lhs, const P &rhs)
{
    P new_pt{};
    new_pt._x = lhs._x + rhs._x;
    new_pt._y = lhs._y + rhs._y;
    return new_pt;
}

P add(const P &lhs, const P &rhs)
{
    P new_pt{};
    new_pt.x(lhs.x() + rhs.x());
    new_pt.y(lhs.y() + rhs.y());
    return new_pt;
}

int main()
{
    P pt1{1.0, 1.0};
    P pt2{2.0, 2.0};

    auto pt3 = pt1 + pt2;
    auto pt4 = add(pt1, pt2);
    return 0;
}
```

```
function __ZplRK1PS1_() {
    xmm0 = intrinsic_movsd(xmm0, *arg0);
    intrinsic_movsd(xmm1, *(arg0 + 0x8));
    intrinsic_addsd(xmm0, *arg1);
    intrinsic_addsd(xmm1, *(arg1 + 0x8));
    return rax;
}

function __Z3addRK1PS1_() {
    xmm0 = intrinsic_movsd(xmm0, *arg0);
    intrinsic_movsd(xmm1, *(arg0 + 0x8));
    intrinsic_addsd(xmm0, *arg1);
    intrinsic_addsd(xmm1, *(arg1 + 0x8));
    return rax;
}
```

```
|-FunctionDecl prev 0x7fa208691130 used operator+ 'class P (const class P &, const class P &)'
| |-ParmVarDecl used lhs 'const class P &'
| |-ParmVarDecl used rhs 'const class P &'
| `-CompoundStmt
|   |-DeclStmt
|   | `-VarDecl used new_pt 'class P' nrvo listinit
|   |   `-CXXConstructExpr 'class P' 'void (void) noexcept' zeroing
|   |-BinaryOperator 'double' lvalue '='
|   | |-MemberExpr 'double' lvalue ._x
|   | | `-DeclRefExpr 'class P' lvalue Var 'new_pt' 'class P'
|   | `-BinaryOperator 'double' '+'
|   |   |-ImplicitCastExpr 'double' <LValueToRValue>
|   |   | `-MemberExpr 'const double' lvalue ._x
|   |   |   `-DeclRefExpr 'const class P' lvalue ParmVar 'lhs' 'const class P &'
|   |   `-ImplicitCastExpr 'double' <LValueToRValue>
|   |     `-MemberExpr 'const double' lvalue ._x
|   |       `-DeclRefExpr 'const class P' lvalue ParmVar 'rhs' 'const class P &'
|   |-BinaryOperator 'double' lvalue '='
|   | |-MemberExpr 'double' lvalue ._y
|   | | `-DeclRefExpr 'class P' lvalue Var 'new_pt' 'class P'
|   | `-BinaryOperator 'double' '+'
|   |   |-ImplicitCastExpr 'double' <LValueToRValue>
|   |   | `-MemberExpr 'const double' lvalue ._y
|   |   |   `-DeclRefExpr 'const class P' lvalue ParmVar 'lhs' 'const class P &'
|   |   `-ImplicitCastExpr 'double' <LValueToRValue>
|   |     `-MemberExpr 'const double' lvalue ._y
|   |       `-DeclRefExpr 'const class P' lvalue ParmVar 'rhs' 'const class P &'
|   `-ReturnStmt
|     `-CXXConstructExpr 'class P' 'void (const class P &) noexcept' elidable
|       `-ImplicitCastExpr 'const class P' lvalue <NoOp>
|         `-DeclRefExpr 'class P' lvalue Var 'new_pt' 'class P'
|-FunctionDecl used add 'class P (const class P &, const class P &)'
| |-ParmVarDecl used lhs 'const class P &'
| |-ParmVarDecl used rhs 'const class P &'
| `-CompoundStmt
|   |-DeclStmt
|   | `-VarDecl used new_pt 'class P' nrvo listinit
|   |   `-CXXConstructExpr 'class P' 'void (void) noexcept' zeroing
|   |-CXXMemberCallExpr 'void'
|   | |-MemberExpr '<bound member function type>' .x
|   | | `-DeclRefExpr 'class P' lvalue Var 0x7fa208692450 'new_pt' 'class P'
|   | `-BinaryOperator 'double' '+'
|   |   |-CXXMemberCallExpr 'double'
|   |   | `-MemberExpr '<bound member function type>' .x
|   |   |   `-DeclRefExpr 'const class P' lvalue ParmVar 'lhs' 'const class P &'
|   |   `-CXXMemberCallExpr 'double'
|   |     `-MemberExpr '<bound member function type>' .x
|   |       `-DeclRefExpr 'const class P' lvalue ParmVar 'rhs' 'const class P &'
|   |-CXXMemberCallExpr 'void'
|   | |-MemberExpr '<bound member function type>' .y
|   | | `-DeclRefExpr 'class P' lvalue Var 0x7fa208692450 'new_pt' 'class P'
|   | `-BinaryOperator 'double' '+'
|   |   |-CXXMemberCallExpr 'double'
|   |   | `-MemberExpr '<bound member function type>' .y
|   |   |   `-DeclRefExpr 'const class P' lvalue ParmVar 'lhs' 'const class P &'
|   |   `-CXXMemberCallExpr 'double'
|   |     `-MemberExpr '<bound member function type>' .y
|   |       `-DeclRefExpr 'const class P' lvalue ParmVar 'rhs' 'const class P &'
|   `-ReturnStmt
|     `-CXXConstructExpr 'class P' 'void (const class P &) noexcept' elidable
|       `-ImplicitCastExpr 'const class P' lvalue <NoOp>
|         `-DeclRefExpr 'class P' lvalue Var 0x7fa208692450 'new_pt' 'class P'
```

Object Copy Semantics

```
A pass_by_value(A a)
{
    return a;
}

A pass_by_ref(A &a)
{
    return a;
}

A pass_by_const_ref(const A &a)
{
    return a;
}
```

```
int main()
{
    A a1{};
    A a2{"string"};
    a1 = a2;
    auto a3 = A{};
    auto a4 = pass_by_value(a3);
    auto a5 = pass_by_ref(a4);
    auto a6 = pass_by_const_ref(a5);
```

```
|-FunctionDecl used pass_by_value 'struct A (struct A)'
| |-ParmVarDecl used a 'struct A'
| `-CompoundStmt
|   `-ReturnStmt
|     `-CXXConstructExpr 'struct A' 'void (struct A &&) noexcept'
|       `-ImplicitCastExpr 'struct A' xvalue <NoOp>
|         `-DeclRefExpr 'struct A' lvalue ParmVar 0x7fd157bbf860 'a' 'struct A'
|-FunctionDecl used pass_by_ref 'struct A (struct A &)'
| |-ParmVarDecl used a 'struct A &'
| `-CompoundStmt
|   `-ReturnStmt
|     `-CXXConstructExpr 'struct A' 'void (const struct A &)'
|       `-ImplicitCastExpr 'const struct A' lvalue <NoOp>
|         `-DeclRefExpr 'struct A' lvalue ParmVar 0x7fd157bbfa80 'a' 'struct A &'
|-FunctionDecl used pass_by_const_ref 'struct A (const struct A &)'
| |-ParmVarDecl used a 'const struct A &'
| `-CompoundStmt
|   `-ReturnStmt
|     `-CXXConstructExpr 'struct A' 'void (const struct A &)'
|       `-DeclRefExpr 'const struct A' lvalue ParmVar 0x7fd157bbfca0 'a' 'const struct A &'


`-FunctionDecl main 'int (void)'
  `-CompoundStmt
    |-DeclStmt
    | `-VarDecl used a1 'struct A' listinit
    |   `-CXXConstructExpr 'struct A' 'void (void)'
    |-DeclStmt
    | `-VarDecl used a2 'struct A' listinit
    |   `-ExprWithCleanups 'struct A'
    |     `-CXXConstructExpr 'struct A' 'void (const std::string &)'
    |       `-MaterializeTemporaryExpr 'const std::string':'const class std::__1::basic_string<char>' lvalue
    |         `-CXXBindTemporaryExpr 'const std::string':'const class std::__1::basic_string<char>' (CXXTemporary 0x7fd157bc4338)
    |           `-CXXConstructExpr 'const std::string':'const class std::__1::basic_string<char>' 'void (const char *)'
    |             `-ImplicitCastExpr 'const char *' <ArrayToPointerDecay>
    |               `-StringLiteral 'const char [7]' lvalue "string"
    |-CXXOperatorCallExpr 'struct A' lvalue
    | |-ImplicitCastExpr 'struct A &(*)(const struct A &)' <FunctionToPointerDecay>
    | | `-DeclRefExpr 'struct A &(const struct A &)' lvalue CXXMethod 0x7fd157bb0020 'operator=' 'struct A &(const struct A &)'
    | |-DeclRefExpr 'struct A' lvalue Var 0x7fd157bc4100 'a1' 'struct A'
    | `-ImplicitCastExpr 'const struct A' lvalue <NoOp>
    |   `-DeclRefExpr 'struct A' lvalue Var 0x7fd157bc4210 'a2' 'struct A'
    |-DeclStmt
    | `-VarDecl used a3 'struct A':'struct A' cinit
    |   `-ExprWithCleanups 'struct A':'struct A'
    |     `-CXXConstructExpr 'struct A':'struct A' 'void (struct A &&) noexcept' elidable
    |       `-MaterializeTemporaryExpr 'struct A' xvalue
    |         `-CXXBindTemporaryExpr 'struct A' (CXXTemporary 0x7fd157bc45f0)
    |           `-CXXTemporaryObjectExpr 'struct A' 'void (void)'
    |-DeclStmt
    | `-VarDecl used a4 'struct A':'struct A' cinit
    |   `-ExprWithCleanups 'struct A':'struct A'
    |     `-CXXConstructExpr 'struct A':'struct A' 'void (struct A &&) noexcept' elidable
    |       `-MaterializeTemporaryExpr 'struct A' xvalue
    |         `-CXXBindTemporaryExpr 'struct A' (CXXTemporary 0x7fd157bc4938)
    |           `-CallExpr 'struct A'
    |             |-ImplicitCastExpr 'struct A (*)(struct A)' <FunctionToPointerDecay>
    |             | `-DeclRefExpr 'struct A (struct A)' lvalue Function 0x7fd157bbf920 'pass_by_value' 'struct A (struct A)'
    |             `-CXXBindTemporaryExpr 'struct A' (CXXTemporary 0x7fd157bc4910)
    |               `-CXXConstructExpr 'struct A' 'void (const struct A &)'
    |                 `-ImplicitCastExpr 'const struct A':'const struct A' lvalue <NoOp>
    |                   `-DeclRefExpr 'struct A':'struct A' lvalue Var 0x7fd157bc4508 'a3' 'struct A':'struct A'
    |-DeclStmt
    | `-VarDecl used a5 'struct A':'struct A' cinit
    |   `-ExprWithCleanups 'struct A':'struct A'
    |     `-CXXConstructExpr 'struct A':'struct A' 'void (struct A &&) noexcept' elidable
    |       `-MaterializeTemporaryExpr 'struct A' xvalue
    |         `-CXXBindTemporaryExpr 'struct A' (CXXTemporary 0x7fd157bc4bd0)
    |           `-CallExpr 'struct A'
    |             |-ImplicitCastExpr 'struct A (*)(struct A &)' <FunctionToPointerDecay>
    |             | `-DeclRefExpr 'struct A (struct A &)' lvalue Function 0x7fd157bbfb40 'pass_by_ref' 'struct A (struct A &)'
    |             `-DeclRefExpr 'struct A':'struct A' lvalue Var 0x7fd157bc4740 'a4' 'struct A':'struct A'
    |-DeclStmt
    | `-VarDecl a6 'struct A':'struct A' cinit
    |   `-ExprWithCleanups 'struct A':'struct A'
    |     `-CXXConstructExpr 'struct A':'struct A' 'void (struct A &&) noexcept' elidable
    |       `-MaterializeTemporaryExpr 'struct A' xvalue
    |         `-CXXBindTemporaryExpr 'struct A' (CXXTemporary 0x7fd157bc4e88)
    |           `-CallExpr 'struct A'
    |             |-ImplicitCastExpr 'struct A (*)(const struct A &)' <FunctionToPointerDecay>
    |             | `-DeclRefExpr 'struct A (const struct A &)' lvalue Function 0x7fd157bbfd60 'pass_by_const_ref' 'struct A (const struct A &)'
    |             `-ImplicitCastExpr 'const struct A':'const struct A' lvalue <NoOp>
    |               `-DeclRefExpr 'struct A':'struct A' lvalue Var 0x7fd157bc4a58 'a5' 'struct A':'struct A'
```

Object Move Semantics

```
A func1(A a)
{
    return a;
}

A func2(A &&a)
{
    return std::move(a);
}

A func3(A &&a)
{
    return a;
}
```

```
int main()
{
    A a1{};
    auto a2 = std::move(a1);

    A a3{};
    a3 = std::move(a1);

    auto a4 = func1(a1);
    auto a5 = func1({});

    auto a6 = func2(std::move(a1));

    A a7{};
    a7 = func2({});

    auto a8 = func3(std::move(a1));

    return 0;
}
```

```
`-FunctionDecl main 'int (void)'
  `-CompoundStmt
    |-DeclStmt
    | `-VarDecl used a1 'struct A' listinit
    |   `-CXXConstructExpr 'struct A' 'void (void)'
    |-DeclStmt
    | `-VarDecl a2 'struct A':'struct A' cinit
    |   `-CXXConstructExpr 'struct A':'struct A' 'void (struct A &&) noexcept'
    |     `-CallExpr 'typename remove_reference<struct A &>::type':'struct A' xvalue
    |       |-ImplicitCastExpr 'typename remove_reference<struct A &>::type &&(*)(struct A &&) noexcept' <FunctionToPointerDecay>
    |       | `-DeclRefExpr 'typename remove_reference<struct A &>::type &&(struct A &&) noexcept' lvalue Function 0x7fe0280b0430
    |       |         'move' 'typename remove_reference<struct A &>::type &&(struct A &&) noexcept' (FunctionTemplate 0x7fe028821408 'move')
    |       `-DeclRefExpr 'struct A' lvalue Var 0x7fe0280b0c10 'a1' 'struct A'
    |-DeclStmt
    | `-VarDecl used a3 'struct A' listinit
    |   `-CXXConstructExpr 'struct A' 'void (void)'
    |-CXXOperatorCallExpr 'struct A' lvalue
    | |-ImplicitCastExpr 'struct A &(*)(struct A &&) noexcept' <FunctionToPointerDecay>
    | | `-DeclRefExpr 'struct A &(struct A &&) noexcept' lvalue CXXMethod 0x7fe0280af618 'operator=' 'struct A &(struct A &&) noexcept'
    | |-DeclRefExpr 'struct A' lvalue Var 0x7fe0280b0fc0 'a3' 'struct A'
    | `-CallExpr 'typename remove_reference<struct A &>::type':'struct A' xvalue
    |   |-ImplicitCastExpr 'typename remove_reference<struct A &>::type &&(*)(struct A &&) noexcept' <FunctionToPointerDecay>
    |   | `-DeclRefExpr 'typename remove_reference<struct A &>::type &&(struct A &&) noexcept' lvalue Function 0x7fe0280b0430
    |   |       'move' 'typename remove_reference<struct A &>::type &&(struct A &&) noexcept' (FunctionTemplate 0x7fe028821408 'move')
    |   `-DeclRefExpr 'struct A' lvalue Var 0x7fe0280b0c10 'a1' 'struct A'
    |-DeclStmt
    | `-VarDecl a4 'struct A':'struct A' cinit
    |   `-ExprWithCleanups 'struct A':'struct A'
    |     `-CXXConstructExpr 'struct A':'struct A' 'void (struct A &&) noexcept' elidable
    |       `-MaterializeTemporaryExpr 'struct A' xvalue
    |         `-CallExpr 'struct A'
    |           |-ImplicitCastExpr 'struct A (*)(struct A)' <FunctionToPointerDecay>
    |           | `-DeclRefExpr 'struct A (struct A)' lvalue Function 0x7fe0280af9e0 'func1' 'struct A (struct A)'
    |           `-CXXConstructExpr 'struct A' 'void (const struct A &)'
    |             `-ImplicitCastExpr 'const struct A' lvalue <NoOp>
    |               `-DeclRefExpr 'struct A' lvalue Var 0x7fe0280b0c10 'a1' 'struct A'
    |-DeclStmt
    | `-VarDecl a5 'struct A':'struct A' cinit
    |   `-ExprWithCleanups 'struct A':'struct A'
    |     `-CXXConstructExpr 'struct A':'struct A' 'void (struct A &&) noexcept' elidable
    |       `-MaterializeTemporaryExpr 'struct A' xvalue
    |         `-CallExpr 'struct A'
    |           |-ImplicitCastExpr 'struct A (*)(struct A)' <FunctionToPointerDecay>
    |           | `-DeclRefExpr 'struct A (struct A)' lvalue Function 0x7fe0280af9e0 'func1' 'struct A (struct A)'
    |           `-CXXConstructExpr 'struct A' 'void (void)'
    |-DeclStmt
    | `-VarDecl a6 'struct A':'struct A' cinit
    |   `-ExprWithCleanups 'struct A':'struct A'
    |     `-CXXConstructExpr 'struct A':'struct A' 'void (struct A &&) noexcept' elidable
    |       `-MaterializeTemporaryExpr 'struct A' xvalue
    |         `-CallExpr 'struct A'
    |           |-ImplicitCastExpr 'struct A (*)(struct A &&)' <FunctionToPointerDecay>
    |           | `-DeclRefExpr 'struct A (struct A &&)' lvalue Function 0x7fe0280afc00 'func2' 'struct A (struct A &&)'
    |           `-CallExpr 'typename remove_reference<struct A &>::type':'struct A' xvalue
    |             |-ImplicitCastExpr 'typename remove_reference<struct A &>::type &&(*)(struct A &&) noexcept' <FunctionToPointerDecay>
    |             | `-DeclRefExpr 'typename remove_reference<struct A &>::type &&(struct A &&) noexcept' lvalue Function 0x7fe0280b0430
    |             |       'move' 'typename remove_reference<struct A &>::type &&(struct A &&) noexcept' (FunctionTemplate 0x7fe028821408 'move')
    |             `-DeclRefExpr 'struct A' lvalue Var 0x7fe0280b0c10 'a1' 'struct A'
    |-DeclStmt
    | `-VarDecl used a7 'struct A' listinit
    |   `-CXXConstructExpr 'struct A' 'void (void)'
    |-ExprWithCleanups 'struct A' lvalue
    | `-CXXOperatorCallExpr 'struct A' lvalue
    |   |-ImplicitCastExpr 'struct A &(*)(struct A &&) noexcept' <FunctionToPointerDecay>
    |   | `-DeclRefExpr 'struct A &(struct A &&) noexcept' lvalue CXXMethod 0x7fe0280af618 'operator=' 'struct A &(struct A &&) noexcept'
    |   |-DeclRefExpr 'struct A' lvalue Var 0x7fe0280b1c28 'a7' 'struct A'
    |   `-MaterializeTemporaryExpr 'struct A' xvalue
    |     `-CallExpr 'struct A'
    |       |-ImplicitCastExpr 'struct A (*)(struct A &&)' <FunctionToPointerDecay>
    |       | `-DeclRefExpr 'struct A (struct A &&)' lvalue Function 0x7fe0280afc00 'func2' 'struct A (struct A &&)'
    |       `-MaterializeTemporaryExpr 'struct A' xvalue
    |         `-CXXConstructExpr 'struct A' 'void (void)'
    |-DeclStmt
    | `-VarDecl a8 'struct A':'struct A' cinit
    |   `-ExprWithCleanups 'struct A':'struct A'
    |     `-CXXConstructExpr 'struct A':'struct A' 'void (struct A &&) noexcept' elidable
    |       `-MaterializeTemporaryExpr 'struct A' xvalue
    |         `-CallExpr 'struct A'
    |           |-ImplicitCastExpr 'struct A (*)(struct A &&)' <FunctionToPointerDecay>
    |           | `-DeclRefExpr 'struct A (struct A &&)' lvalue Function 0x7fe0280b09b8 'func3' 'struct A (struct A &&)'
    |           `-CallExpr 'typename remove_reference<struct A &>::type':'struct A' xvalue
    |             |-ImplicitCastExpr 'typename remove_reference<struct A &>::type &&(*)(struct A &&) noexcept' <FunctionToPointerDecay>
    |             | `-DeclRefExpr 'typename remove_reference<struct A &>::type &&(struct A &&) noexcept' lvalue Function 0x7fe0280b0430
    |             |       ‘move' 'typename remove_reference<struct A &>::type &&(struct A &&) noexcept' (FunctionTemplate 0x7fe028821408 'move')
    |             `-DeclRefExpr 'struct A' lvalue Var 0x7fe0280b0c10 'a1' 'struct A'
    |
    `-ReturnStmt
      `-IntegerLiteral 'int' 0
```

Object Destruction

```
enum class Selector { a, b, c };

void func1(Selector v)
{
    P a{0., 0., 0.};

    switch (v) {
        case Selector::a:
            return;
        case Selector::b:
            return;
        case Selector::c:
            return;
    }
}
```

```
int main()
{
    {
        P a{0., 0., 0.};
    }
    func1(Selector::a);
    func1(Selector::b);
    func1(Selector::c);
    return 0;
}
```

```
void func1Selector(int arg0)
{
    P a;
    PConstructor(&a, 0x0, 0x0, 0x0);
    int __temp0;
    if (arg0 != 0x0) {
        if (arg0 - 0x1 != 0x0) {
            if (arg0 - 0x2 != 0x0) {
                __temp0 = 0x0;
            } else {
                __temp0 = 0x1;
            }
        } else {
            __temp0 = 0x1;
        }
    } else {
        __temp0 = 0x1;
    }
    PDestructor(&a); // call destructor on all paths
    if (__temp0 - 0x1 <= 0x0) {
        // pass
    } else {
        // reconstruct if arg0 is not a Selector value
        PConstructor(&a, 0x0, 0x0, 0x0);
    }
}
```

```
int main()
{
    P a;
    PConstructor(&a, 0x0, 0x0, 0x0);
    PDestructor(&a);
    func1Selector(0x0);
    func1Selector(0x1);
    func1Selector(0x2);
    return 0x0;
}
```

Arrays of Objects

```
struct P {
    double x;
    double y;
};

int main()
{
    P pa1[10] = {P{}, P{1.0, 1.0}, static_cast<double>(-1)};
    P pa2[10];
    return 0;
}
```

```
#include <memory.h>
#include <printf.h>

#define GUARD_MAX 8

// randomly generated
long __stack_chk_guard[GUARD_MAX] = {
        3673554329674219293,
        7651880036576883891,
        -6765366229887973090,
        2894016341385819420,
        -10841679571938245 86,
        -8948618946371855535,
        1882061842436483468,
        -8518030012397825452
};

// mock void __stack_chk_fail() from libc/sys/OpenBSD/
stack_protector.c
int __stack_chk_fail()
{
    fprintf(stderr, "[PID] stack overflow\n");
    // abort();
    return 0xFFFFFFFF;
}

typedef struct _P {
    double x;
    double y;
} P;

int main()
{
    // optimized out by compiler but used to show purpose
of __stack_chk_guard
    long *___stack_chk_guard = __stack_chk_guard;
    double __temp0 = -1.;
    double __temp1 = 1.;
    P pa1[10];
    memset(&pa1, 0x0, 0xa0);
    *(double *) ((void *) pa1 + 0x10) = __temp1;
    *(double *) ((void *) pa1 + 0x18) = __temp1;
    *(double *) ((void *) pa1 + 0x20) = __temp0;
    int rax;
    if (*___stack_chk_guard == *___stack_chk_guard) {
        rax = 0x0;
    } else {
        rax = __stack_chk_fail();
    }
    return rax;
}
```

```
`-FunctionDecl main 'int (void)'
  `-CompoundStmt
    |-DeclStmt
    | `-VarDecl pa1 'struct P [10]' cinit
    |   `-ExprWithCleanups 'struct P [10]'
    |     `-InitListExpr 'struct P [10]'
    |       |-array filler
    |       | `-InitListExpr 'struct P'
    |       |   |-ImplicitValueInitExpr 'double'
    |       |   `-ImplicitValueInitExpr 'double'
    |       |-CXXConstructExpr 'struct P' 'void (struct P &&) noexcept' elidable
    |       | `-MaterializeTemporaryExpr 'struct P' xvalue
    |       |   `-CXXFunctionalCastExpr 'struct P' functional cast to struct P <NoOp>
    |       |     `-InitListExpr 'struct P'
    |       |       |-ImplicitValueInitExpr 'double'
    |       |       `-ImplicitValueInitExpr 'double'
    |       |-CXXConstructExpr 'struct P' 'void (struct P &&) noexcept' elidable
    |       | `-MaterializeTemporaryExpr 'struct P' xvalue
    |       |   `-CXXFunctionalCastExpr 'struct P' functional cast to struct P <NoOp>
    |       |     `-InitListExpr 'struct P'
    |       |       |-FloatingLiteral 'double' 1.000000e+00
    |       |       `-FloatingLiteral 'double' 1.000000e+00
    |       `-InitListExpr 'struct P'
    |         |-CXXStaticCastExpr 'double' static_cast<double> <NoOp>
    |         | `-ImplicitCastExpr 'double' <IntegralToFloating>
    |         |   `-UnaryOperator 'int' prefix '-'
    |         |     `-IntegerLiteral 'int' 1
    |         `-ImplicitValueInitExpr 'double'
    |-DeclStmt
    | `-VarDecl pa2 'struct P [10]' callinit
    |   `-CXXConstructExpr 'struct P [10]' 'void (void) noexcept'
    `-ReturnStmt
      `-IntegerLiteral 'int' 0
```

__stack_chk_guard

- __stack_chk_guard (stack protector)
  - a compiler feature that helps to detect stack buffer overrun by aborting if a compiler-inserted value on the stack is changed (only detects, does not prevent)
  - also helps to mitigate return-oriented programming exploits
- __stack_chk_fail is a function that logs stack overflow and then calls abort
- these helpers are compiled into libc and then pulled into libSystem.B.dylib from the Libsystem scripts
- https://opensource.apple.com/source/Libsystem/
- also in libkern/stack_protector.c

- GCC stack protector flags:
  - -fstack-protector - check for stack smashing in functions with vulnerable objects
  - Includes functions with buffers larger than 8 bytes or calls to alloca
- -fstack-protector-strong - also includes functions with local arrays or references to local frame addresses
- -fstack-protector-all - checks for stack smashing in every function
- -fstack-shuffle - (OpenBSD) randomizes the order of stack variables at compile time

Operators new and delete

```c
#include <memory.h>
#include <stdlib.h>
#include <stdio.h>

typedef struct _P3 {
    double x;
    double y;
    double z;
} P3;

void __cxa_new_handler()
{
    fprintf(stderr, "bad malloc");
    abort();
}

void *get_new_handler()
{
    // synchronize access with mfence
    instruction
    return (void *) __cxa_new_handler;
}

// see libcxxabi/src/cxa_exception.cpp
// void __cxa_throw(void *thrown_object,
std::type_info *tinfo, void (*dest)(void *));

void __cxa_throw()
{
    fprintf(stderr, "__cxa_throw");
    abort();
}

void *operator_new(unsigned long arg0)
{
    void *eax;
    unsigned long edi = 0x1;
    if (arg0 != 0x0) {
        edi = arg0;
    }
    goto loc_a;

    loc_a:
    eax = malloc(edi);
    if (eax != 0x0) {
        goto loc_d;
    }

    loc_b:
    eax = get_new_handler();
    if (eax != 0x0) {
        goto loc_e;
    }

    loc_c:
    __cxa_throw();
    return eax;

    loc_d:
    return eax;

    loc_e:
    ((void (*)()) eax)();
    goto loc_a;
}

void operator_delete(void *arg0)
{
    void *__temp1 = arg0;
    if (__temp1 != NULL) {
        free(arg0);
    }
}

int main()
{
    void *__temp0 = operator_new(0x4);
    *(int32_t *) __temp0 = 0x5;
    int32_t *pi = __temp0;
    if (__temp0 != NULL) {
        operator_delete(pi);
    }

    void *__temp1 = operator_new(0x18);
    memset(__temp1, 0x0, 0x18);
    P3 *pp = __temp1;
    if (__temp1 != NULL) {
        operator_delete(pp);
    }
    return 0x0;
}
```

```c
struct P3 {
    double x;
    double y;
    double z;
};

int main()
{
    auto pi = new int{5};
    delete pi;

    auto pp = new P3{};
    delete pp;

    return 0;
}
```

```
|-FunctionDecl main 'int (void)'
| `-CompoundStmt
|   |-DeclStmt
|   | `-VarDecl used pi 'int *':'int *' cinit
|   |   `-CXXNewExpr 'int *' Function 'operator new' 'void *(unsigned long)'
|   |     `-InitListExpr 'int'
|   |       `-IntegerLiteral 'int' 5
|   |-CXXDeleteExpr 'void' Function 'operator delete' 'void (void *) noexcept'
|   | `-ImplicitCastExpr 'int *':'int *' <LValueToRValue>
|   |   `-DeclRefExpr 'int *':'int *' lvalue Var 'pi' 'int *':'int *'
|   |-DeclStmt
|   | `-VarDecl used pp 'struct P3 *':'struct P3 *' cinit
|   |   `-CXXNewExpr 'struct P3 *' Function 'operator new' 'void *(unsigned long)'
|   |     `-InitListExpr 'struct P3'
|   |       |-ImplicitValueInitExpr 'double'
|   |       |-ImplicitValueInitExpr 'double'
|   |       `-ImplicitValueInitExpr 'double'
|   |-CXXDeleteExpr 'void' Function 'operator delete' 'void (void *) noexcept'
|   | `-ImplicitCastExpr 'struct P3 *':'struct P3 *' <LValueToRValue>
|   |   `-DeclRefExpr 'struct P3 *':'struct P3 *' lvalue Var 'pp'
|                 'struct P3 *':'struct P3 *'
|   `-ReturnStmt
|     `-IntegerLiteral 'int' 0
|-FunctionDecl implicit used operator new 'void *(unsigned long)'
| |-ParmVarDecl implicit 'unsigned long'
| `-VisibilityAttr Implicit Default
|-FunctionDecl implicit operator new[] 'void *(unsigned long)'
| |-ParmVarDecl implicit 'unsigned long'
| `-VisibilityAttr Implicit Default
|-FunctionDecl implicit used operator delete 'void (void *) noexcept'
| |-ParmVarDecl implicit 'void *'
| `-VisibilityAttr Implicit Default
`-FunctionDecl implicit operator delete[] 'void (void *) noexcept'
  |-ParmVarDecl implicit 'void *'
  `-VisibilityAttr Implicit Default
```

Temporary Objects

```
int _main() {
    T::T(&var_8);
    T::T(&var_10);
    operator+(&var_20, &var_8);
    operator-(&var_28, &var_8);
    T::~T();
    var_38 = var_8;
    var_40 = var_10;
    operator*(&var_30, &var_38);
    T::~T();
    T::~T();
    T::~T();
    T::~T();
    T::~T();
    T::~T();
    rax = 0x0;
    return rax;
}
```

```
T operator+(const T &lhs, const T &rhs)
{
    T t{};
    t.i = lhs.i + rhs.i;
    return t;
}

T operator-(T &lhs, T &rhs)
{
    T t{};
    t.i = lhs.i - rhs.i;
    return t;
}

T operator*(T lhs, T rhs)
{
    T t{};
    t.i = lhs.i * rhs.i;
    return t;
}
```

```
int main() {
    T t1{1};
    T t2{2};
    auto t = t1 + t2;
    t = t1 - t2;
    t = t1 * t2;
    return 0;
}
```

```
`-FunctionDecl main 'int (void)'
  `-CompoundStmt
    |-DeclStmt
    | `-VarDecl used t1 'struct T' listinit
    |   `-CXXConstructExpr 'struct T' 'void (int)'
    |     `-IntegerLiteral 'int' 1
    |-DeclStmt
    | `-VarDecl used t2 'struct T' listinit
    |   `-CXXConstructExpr 'struct T' 'void (int)'
    |     `-IntegerLiteral 'int' 2
    |-DeclStmt
    | `-VarDecl used t 'struct T':'struct T' cinit
    |   `-ExprWithCleanups 'struct T':'struct T'
    |     `-CXXConstructExpr 'struct T':'struct T' 'void (const struct T &) noexcept' elidable
    |       `-MaterializeTemporaryExpr 'const struct T' lvalue
    |         `-ImplicitCastExpr 'const struct T' <NoOp>
    |           `-CXXBindTemporaryExpr 'struct T' (CXXTemporary 0x7fefd8830f38)
    |             `-CXXOperatorCallExpr 'struct T'
    |               |-ImplicitCastExpr 'struct T (*)(const struct T &, const struct T &)'
<FunctionToPointerDecay>
    |               | `-DeclRefExpr 'struct T (const struct T &, const struct T &)' lvalue Function
0x7fefd8800f70 'operator+' 'struct T (const struct T &, const struct T &)'
    |               |-ImplicitCastExpr 'const struct T' lvalue <NoOp>
    |               | `-DeclRefExpr 'struct T' lvalue Var 0x7fefd8830b18 't1' 'struct T'
    |               `-ImplicitCastExpr 'const struct T' lvalue <NoOp>
    |                 `-DeclRefExpr 'struct T' lvalue Var 0x7fefd8830c40 't2' 'struct T'
    |-ExprWithCleanups 'struct T' lvalue
    | `-CXXOperatorCallExpr 'struct T' lvalue
    |   |-ImplicitCastExpr 'struct T &(*)(const struct T &) noexcept' <FunctionToPointerDecay>
    |   | `-DeclRefExpr 'struct T &(const struct T &) noexcept' lvalue CXXMethod 0x7fefd8831210 'operator='
'struct T &(const struct T &) noexcept'
    |   |-DeclRefExpr 'struct T':'struct T' lvalue Var 0x7fefd8830d90 't' 'struct T':'struct T'
    |   `-MaterializeTemporaryExpr 'const struct T' lvalue
    |     `-ImplicitCastExpr 'const struct T' <NoOp>
    |       `-CXXBindTemporaryExpr 'struct T' (CXXTemporary 0x7fefd88311e8)
    |         `-CXXOperatorCallExpr 'struct T'
    |           |-ImplicitCastExpr 'struct T (*)(struct T &, struct T &)' <FunctionToPointerDecay>
    |           | `-DeclRefExpr 'struct T (struct T &, struct T &)' lvalue Function 0x7fefd88300a0 'operator-'
'struct T (struct T &, struct T &)'
    |           |-DeclRefExpr 'struct T' lvalue Var 0x7fefd8830b18 't1' 'struct T'
    |           `-DeclRefExpr 'struct T' lvalue Var 0x7fefd8830c40 't2' 'struct T'
    |-ExprWithCleanups 'struct T' lvalue
    | `-CXXOperatorCallExpr 'struct T' lvalue
    |   |-ImplicitCastExpr 'struct T &(*)(const struct T &) noexcept' <FunctionToPointerDecay>
    |   | `-DeclRefExpr 'struct T &(const struct T &) noexcept' lvalue CXXMethod 0x7fefd8831210 'operator='
  'struct T &(const struct T &) noexcept'
    |   |-DeclRefExpr 'struct T':'struct T' lvalue Var 0x7fefd8830d90 't' 'struct T':'struct T'
    |   `-MaterializeTemporaryExpr 'const struct T' lvalue
    |     `-ImplicitCastExpr 'const struct T' <NoOp>
    |       `-CXXBindTemporaryExpr 'struct T' (CXXTemporary 0x7fefd8831888)
    |         `-CXXOperatorCallExpr 'struct T'
    |           |-ImplicitCastExpr 'struct T (*)(struct T, struct T)' <FunctionToPointerDecay>
    |           | `-DeclRefExpr 'struct T (struct T, struct T)' lvalue Function 0x7fefd88305f0 'operator*'
  'struct T (struct T, struct T)'
    |           |-CXXBindTemporaryExpr 'struct T' (CXXTemporary 0x7fefd8831738)
    |           | `-CXXConstructExpr 'struct T' 'void (const struct T &) noexcept'
    |           |   `-ImplicitCastExpr 'const struct T' lvalue <NoOp>
    |           |     `-DeclRefExpr 'struct T' lvalue Var 0x7fefd8830b18 't1' 'struct T'
    |           `-CXXBindTemporaryExpr 'struct T' (CXXTemporary 0x7fefd88317b0)
    |             `-CXXConstructExpr 'struct T' 'void (const struct T &) noexcept'
    |               `-ImplicitCastExpr 'const struct T' lvalue <NoOp>
    |                 `-DeclRefExpr 'struct T' lvalue Var 0x7fefd8830c40 't2' 'struct T'
    `-ReturnStmt
      `-IntegerLiteral 'int' 0
```

Templates

```
`nm -an 01_templates | c++filt`
                 U dyld_stub_binder
0000000100000000 T __mh_execute_header
0000000100000ec0 T A<int>::m()
0000000100000ed0 T A<double>::m()
0000000100000ee0 T void f<double>(double)
0000000100000ef0 T void f<int>(int)
0000000100000f00 T _main
0000000100000f60 T void B::operator()<double>(double
const&)
0000000100000f70 T void B::operator()<int>(int const&)
```

```
|-ClassTemplateDecl A
| |-TemplateTypeParmDecl typename depth 0 index 0 T
| |-CXXRecordDecl class A definition
| | |-CXXRecordDecl implicit class A
| | |-AccessSpecDecl public
| | `-CXXMethodDecl m 'void (void)'
| |   `-CompoundStmt
| |-ClassTemplateSpecialization 'A'
| `-ClassTemplateSpecializationDecl class A definition
|   |-TemplateArgument type 'double'
|   |-CXXRecordDecl prev 0x7f9c858e8460 implicit class A
|   |-AccessSpecDecl public
|   `-CXXMethodDecl m 'void (void)'
|     `-CompoundStmt
|-ClassTemplateSpecializationDecl class A definition
| |-TemplateArgument type 'int'
| |-CXXRecordDecl prev 0x7f9c858e80c0 implicit class A
| |-AccessSpecDecl public
| `-CXXMethodDecl m 'void (void)'
|   `-CompoundStmt
|-FunctionTemplateDecl f
| |-TemplateTypeParmDecl referenced typename depth 0 index 0 T
| |-FunctionDecl f 'void (T)'
| | `-ParmVarDecl s 'T'
| |-FunctionDecl f 'void (double)'
| | |-TemplateArgument type 'double'
| | |-ParmVarDecl s 'double':'double'
| | `-CompoundStmt
| `-FunctionDecl f 'void (int)'
|   |-TemplateArgument type 'int'
|   |-ParmVarDecl s 'int':'int'
|   `-CompoundStmt
|-FunctionTemplateDecl prev 0x7f9c858e8a38 f
| |-TemplateTypeParmDecl referenced typename depth 0 index 0 T
| |-FunctionDecl prev 0x7f9c858e89a0 f 'void (T)'
| | |-ParmVarDecl s 'T'
| | `-CompoundStmt
| |-Function 'f' 'void (double)'
| `-Function 'f' 'void (int)'
|-CXXRecordDecl referenced struct B definition
| |-CXXRecordDecl implicit struct B
| `-FunctionTemplateDecl operator()
|   |-TemplateTypeParmDecl referenced typename depth 0 index 0 T
|   |-CXXMethodDecl operator() 'void (const T &)'
|   | |-ParmVarDecl obj 'const T &'
|   | `-CompoundStmt
|   |-CXXMethodDecl used operator() 'void (const double &)'
|   | |-TemplateArgument type 'double'
|   | |-ParmVarDecl obj 'const double &'
|   | `-CompoundStmt
|   `-CXXMethodDecl used operator() 'void (const int &)'
|     |-TemplateArgument type 'int'
|     |-ParmVarDecl obj 'const int &'
|     `-CompoundStmt
|-ClassTemplateDecl is_void
| |-TemplateTypeParmDecl typename depth 0 index 0 T
| |-CXXRecordDecl struct is_void definition
| | |-public 'std::false_type':'struct std::__1::integral_constant<_Bool, false>'
| | `-CXXRecordDecl implicit struct is_void
| |-ClassTemplateSpecialization 'is_void'
| `-ClassTemplateSpecializationDecl struct is_void definition
|   |-public 'std::false_type':'struct std::__1::integral_constant<_Bool, false>'
|   |-TemplateArgument type 'char'
|   `-CXXRecordDecl prev 0x7f9c858eb0e8 implicit struct is_void
|-ClassTemplateSpecializationDecl struct is_void definition
| |-public 'std::true_type':'struct std::__1::integral_constant<_Bool, true>'
| |-TemplateArgument type 'void'
| `-CXXRecordDecl implicit struct is_void
```

```cpp
#include <type_traits>

template<typename T>
class A {
public:
    void m() {}
};

template
class A<int>;
template void A<double>::m();

template<typename T>
void f(T s);

template<typename T>
void f(T s) {}

template void f<double>(double);
template void f(int);

struct B {
    template<typename T>
    void operator()(const T &obj) {}
};

template<typename T>
struct is_void : std::false_type {};

template<>
struct is_void<void> : std::true_type {};

int main()
{
    A<int> ai{};
    B b{};
    b(1.0);
    b(1);
    return 0;
}
```

Runtime Type Identification

- Information about types should be available at run time for three purposes
  - to support the typeid operator
  - to match an exception handler with a thrown object
  - to implement the dynamic_cast operator
- It is intended that two type_info pointers point to equivalent type descriptions if and only if the pointers are equal. After linking and loading, only one std::type_info structure is accessible via the external name defined by this ABI for any particular complete type symbol

type_info Layout

- The memory layout of a type_info object must follow class layout rules for the host ABI
- type_info objects specify data members of the types they describe (not member functions) but maintain the standard type_info specified API
- Data member layout of type_info objects
  - virtual tables contain one entry describing the offset from a virtual pointer for that virtual table to the origin of the object
    - used for dynamic casts
    - 2 words ahead of the virtual pointer
    - present in all virtual tables, even for classes having virtual bases but no virtual functions.
  - virtual tables contain one entry that is a pointer to an object derived from std::type_info
    - for classes having virtual bases but no virtual functions, the entry is zero
    - the type_info object for a type has a private member, const char *__type_name, that represents the mangled name
    - of the type
    - there are types that are derived from type_info (e.g. __class_type_info) that have additional ABI requirements

type_info Interface

```
namespace std {
    class type_info {
    public:
        virtual ~type_info();
        bool operator==(const type_info& rhs) const noexcept;
        bool operator!=(const type_info& rhs) const noexcept;
        bool before(const type_info& rhs) const noexcept;
        size_t hash_code() const noexcept;
        const char* name() const noexcept;
        type_info(const type_info& rhs) = delete;
        type_info& operator=(const type_info& rhs) = delete;
    };
}
```

```
#include <typeinfo>
#include <string>

struct B {};
struct Derived : B {};

struct B2 {
    virtual void foo() {}
};

struct Derived2 : B2 {};

int main()
{
    auto i = 50;
    std::string s = "string";
    double *pd = nullptr;

    const std::type_info &ti = typeid(i);
    const auto ti_hash = ti.hash_code();
    const auto ti_name = ti.name();

    const std::type_info &ts = typeid(s);
    const auto ts_hash = ts.hash_code();
    const auto ts_name = ts.name();

    const std::type_info &tpd = typeid(pd);
    const auto tpd_hash = tpd.hash_code();
    const auto tpd_name = tpd.name();

    // dereferencing a null pointer OK for non-polymorphic type
    const std::type_info &td = typeid(*pd);
    const auto td_hash = td.hash_code();
    const auto td_name = td.name();

    Derived d1{};
    B &b1 = d1;
    const std::type_info &td1 = typeid(b1);
    const auto td1_hash = td1.hash_code();
    const auto td1_name = td1.name();

    Derived2 d2{};
    B2 &b2 = d2;
    const std::type_info &td2 = typeid(b2);
    const auto td2_hash = td2.hash_code();
    const auto td2_name = td2.name();
    return 0;
}
```

Dynamic Cast

- Overview of types of dynamic casts for polymorphic class types (using the dynamic_cast operator):
  - dynamic_cast<void cv*>, which returns a pointer to the complete lvalue,
  - dynamic_cast operation from a proper base class to a derived class, and
  - dynamic_cast across the hierarchy which can be seen as a cast to the complete lvalue and back to a sibling base.
  - NOTE: some kinds of dynamic_cast operations are handled at compile time and do not need any RTTI
- The most common dynamic cast is base-to-derived in a singly inherited hierarchy

```
extern "C" _LIBCXXABI_FUNC_VIS
void * __dynamic_cast(
    const void *static_ptr,
    const __class_type_info *static_type,
    const __class_type_info *dst_type,
    std::ptrdiff_t src2dst_offset
)
```

  - libcxxabi/src/private_typeinfo.cpp
  - static_ptr - a pointer to an object of type static_type (&v in the expression dynamic_cast<T>(v))
    - nonnull
    - since the object is polymorphic, *(void**)static_ptr is a virtual table pointer
  - static_type - static type of the object pointed to by static_ptr
    - guaranteed to be a polymorphic type
  - dst_type - destination type of the cast (the "T" in "dynamic_cast<T>(v)").
  - src2dst_offset - a static hint about the location of the source subobject with respect to the complete object (e.g. not a public base, multiple but never virtual public base, unique public nonvirtual)
    - default - unique public nonvirtual base at offset src2dst_offset from the origin of dst_type
  - (dynamic_ptr, dynamic_type) are the run time type of the complete object referred to by static_ptr and a pointer to it and can be found from static_ptr for polymorphic types
  - (dynamic_ptr, dynamic_type) is the root of a DAG that grows upward
    - Each node of the tree represents a base class/object of its parent (or parents) below
    - Each node is uniquely represented by a pointer to the object, and a pointer to a type_info object - its type
    - Different nodes may have the same pointer and different nodes may have the same type
    - Only one node has a specific (pointer-value, type) pair
    - In C++ two objects of the same type can not share the same address
  - Return value/type logic:

```
If there is exactly one dst_type of flavor 1, and
   If there is a public path from that dst_type to (static_ptr, static_type), or
   If there are 0 dst_types of flavor 2, and there is a public path from
       (dynamic_ptr, dynamic_type) to (static_ptr, static_type) and a public
       path from (dynamic_ptr, dynamic_type) to the one dst_type, then return
       a pointer to that dst_type.
Else if there are 0 dst_types of flavor 1 and exactly 1 dst_type of flavor 2, and
   if there is a public path from (dynamic_ptr, dynamic_type) to
   (static_ptr, static_type) and a public path from (dynamic_ptr, dynamic_type)
   to the one dst_type, then return a pointer to that one dst_type.
Else return nullptr.

If dynamic_type == dst_type, then the above algorithm collapses to the
following cheaper algorithm:

If there is a public path from (dynamic_ptr, dynamic_type) to
   (static_ptr, static_type), then return dynamic_ptr.
Else return nullptr.
```

Exceptions vs. Traps vs. Interrupts

- Interrupts and exceptions are events that indicate that a condition exists somewhere in the system, the processor, or within the currently executing program or task that requires the attention of a processor
  - Typically result in a forced transfer of execution from the currently running program or task to a special software routine or task (interrupt handler or an exception handler)
- An interrupt is an asynchronous event that is typically triggered by an I/O device
- An exception is a synchronous event that is generated when the processor detects one or more predefined conditions while executing an instruction
- The IA-32 architecture specifies three classes of exceptions:
  - faults - an exception that can generally be corrected and that, once corrected, allows the program to be restarted with no loss of continuity
  - Trap - an exception that is reported immediately following the execution of the trapping instruction
    - allow execution of a program or task to be continued without loss of program continuity
    - the return address for the trap handler points to the instruction to be executed after the trapping instruction
    - aborts - an exception that does not always report the precise location of the instruction causing the exception and does not allow a restart of the program or task that caused the exception
      - used to report severe errors, such as hardware errors and inconsistent or illegal values in system tables

Exceptions

- Each thread in a C++ program has access to an object of the following class (maintained on a per-thread basis):

```
struct __cxa_eh_globals {
  __cxa_exception *caughtExceptions;
  unsigned int uncaughtExceptions;
};
```

- The fields of this structure are defined as follows:
  - caughtExceptions - field is a list of the active exceptions, organized as a stack with the most recent first
  - uncaughtExceptions - field is a count of uncaught exceptions, for use by the C++ library uncaught_exceptions() routine
- The structure can be obtained by using:

```
__cxa_eh_globals *__cxa_get_globals(void)
```

  - Returns a pointer to the __cxa_eh_globals structure for the current thread, initializing it if necessary

```
__cxa_eh_globals *__cxa_get_globals_fast(void)
```

  - Returns a pointer to the __cxa_eh_globals structure for the current thread, assuming that at least one prior call to __cxa_get_globals has been made from the current thread

Throwing the Exception Object

- After constructing the exception object with the throw argument value, the generated code calls the __cxa_throw runtime library routine (never returns)

```
void __cxa_throw (void *thrown_exception, std::type_info *tinfo, void (*dest) (void *));
```

  - thrown_exception - the address of the thrown exception object (which points to the throw value, after the header)
  - tinfo - a std::type_info pointer with the static type of the throw argument as a std::type_info pointer
    - used for matching potential catch sites to the thrown exception.
  - dest - a destructor pointer to be used eventually to destroy the object
- The __cxa_throw:
  - Obtains the __cxa_exception header from the thrown exception object address

```
__cxa_exception *header = ((__cxa_exception *) thrown_exception - 1);
```

  - Saves the current unexpected_handler and terminate_handler in the __cxa_exception header
  - Saves the tinfo and dest arguments in the __cxa_exception header
  - Sets the exception_class field in the unwind header
  - Increments the uncaught_exception flag
  - Calls _Unwind_RaiseException in the system unwind library with the pointer to the thrown exception as an argument
    - passed to __cxa_throw as an argument
  - __Unwind_RaiseException begins the process of stack unwinding
  - In special cases (e.g. when no handler is found) _Unwind_RaiseException may return
    - __cxa_throw will call terminate if no handler for the exception

Exceptions

```cpp
#ifdef USE_EXCEPTIONS
#include <exception>
#endif

// #define USE_EXCEPTIONS

int func1(bool b);

#ifdef USE_EXCEPTIONS
struct Error_t : public std::exception {
    const char *what() const noexcept { return "Error_t\n"; };
};
#endif

void func2(bool b)
{
#ifdef USE_EXCEPTIONS
    try {
#endif
        auto result = func1(b);
#ifdef USE_EXCEPTIONS
    }
    catch (Error_t &e) {
        const auto w = e.what();
    }
#endif
}

int func1(bool b)
{
    if (b) {
        return 10;
    } else {
#ifdef USE_EXCEPTIONS
        throw Error_t{};
#else
        return -1;
#endif
    }
}

int main()
{
    func2(true);
    func2(false);
    return 0;
}
```

```
size 02_exceptions
__TEXT   __DATA   __OBJC   others   dec   hex
8192  4096  0  4294971392  4294983680  100004000

size -x -l -m 02_exceptions
Segment __PAGEZERO: 0x100000000 (vmaddr 0x0 fileoff 0)
Segment __TEXT: 0x2000 (vmaddr 0x100000000 fileoff 0)
   Section __text: 0x20b (addr 0x100001ce0 offset 7392)
   Section __stubs: 0x2a (addr 0x100001eec offset 7916)
   Section __stub_helper: 0x4c (addr 0x100001f18 offset 7960)
   Section __gcc_except_tab: 0x28 (addr 0x100001f64 offset 8036)
   Section __const: 0x9 (addr 0x100001f8c offset 8076)
   Section __cstring: 0x9 (addr 0x100001f95 offset 8085)
   Section __unwind_info: 0x5c (addr 0x100001fa0 offset 8096)
   total 0x317
Segment __DATA: 0x1000 (vmaddr 0x100002000 fileoff 8192)
   Section __nl_symbol_ptr: 0x10 (addr 0x100002000 offset 8192)
   Section __got: 0x28 (addr 0x100002010 offset 8208)
   Section __la_symbol_ptr: 0x38 (addr 0x100002038 offset 8248)
   Section __data: 0x40 (addr 0x100002070 offset 8304)
   total 0xb0
Segment __LINKEDIT: 0x1000 (vmaddr 0x100003000 fileoff 12288)
total 0x100004000

size 02_exceptions_no_except
__TEXT   __DATA   __OBJC   others   dec   hex
4096  0  0  4294971392 4294975488 100002000

size -x -l -m 02_exceptions_no_except
Segment __PAGEZERO: 0x100000000 (vmaddr 0x0 fileoff 0)
Segment __TEXT: 0x1000 (vmaddr 0x100000000 fileoff 0)
   Section __text: 0x88 (addr 0x100000f30 offset 3888)
   Section __unwind_info: 0x48 (addr 0x100000fb8 offset 4024)
   total 0xd0
Segment __LINKEDIT: 0x1000 (vmaddr 0x100001000 fileoff 4096)
total 0x100002000
```

Unwinding

- The unwinding library interface:
  - _Unwind_RaiseException
  - _Unwind_Resume
  - _Unwind_DeleteException
  - _Unwind_GetGR
  - _Unwind_SetGR
  - _Unwind_GetIP
  - _Unwind_SetIP
  - _Unwind_GetRegionStart
  - _Unwind_GetLanguageSpecificData
  - _Unwind_ForcedUnwind
- Supporting data structures
  - _Unwind_Context
  - _Unwind_Exception
- libunwind.dylib on the Mac
  - src/CompactUnwinder.hpp
  - src/UnwindLevel1.c
- a language and vendor specific routine is stored by the compiler in the unwind descriptor for the stack frames requiring exception processing (personality function)
- The routine is called by the unwinder to handle language-specific tasks such as identifying the frame handling for a particular exception

Reasons for Unwinding

- exceptions, as defined by languages that support them (such as C++)
- "forced" unwinding (such as caused by longjmp or thread termination)
- For exceptions, the stack is unwound while the exception propagates
  - The handler routine for each stack needs to know whether it wants to catch the exception or pass it through
- During "forced unwinding" an external agent is driving the unwinding

The Unwind Process

- The standard ABI exception handling/unwind process begins with the raising of an exception which specifies an exception object and an exception class
- The runtime framework then starts a two-phase process:
  - the search phase - the framework repeatedly calls the handler routine, first for the
  - current instruction and register state, and then unwinding a frame to a new instruction at each step, until the personality routine reports either success (a handler was found in the queried frame) or failure (no handler was found) in all frames
- If the search phase reports failure, e.g. because no handler was found, it will call terminate() rather than moving on to phase 2
- If the search phase reports success, the framework restarts in the cleanup phase
  - it repeatedly calls the personality routine, first for the current instruction and register state, and then unwinding a frame to a new instruction at each step until it gets to the frame with an identified handler
  - Then it restores the register state and control is transferred to the user landing pad code

Compact Unwind Encoding

- https://opensource.apple.com/source/libunwind/libunwind-35.3/include/mach-o/compact_unwind_encoding.h

- The linker creates a __TEXT,__unwind_info section when it creates a final linked image
  - This is a small and fast way for the runtime to access unwind info for any given function
- If the compiler emitted compact unwind info for the function, that compact unwind info will be encoded in the __TEXT,__unwind_info section
- If the compiler emitted dwarf unwind info, the __TEXT,__unwind_info section will contain the offset of the FDE in the __TEXT,__eh_frame section in the final linked image
- Previously, the linker would transform some dwarf unwind info into compact unwind info (fragile and no longer done)
- The compact unwind encoding is a 32-bit value which is encoded in an architecture specific way
- It contains which registers to restore from where and how to unwind out of the function
- Encoding for x86_64:
  - 1-bit: start
  - 1-bit: has lsda
  - 2-bit: personality index
  - 4-bits: 0=old, 1=rbp based, 2=stack-imm, 3=stack-ind, 4=dwarf
  - rbp based:
    - 15-bits (5*3-bits per reg) register permutation
    - 8-bits for stack offset
  - frameless:
    - 8-bits stack size
    - 3-bits stack adjust
    - 3-bits register count
    - 10-bits register permutation

- For x86_64 there are four modes for the compact unwind encoding:
  - UNWIND_X86_64_MODE_RBP_FRAME
  - UNWIND_X86_64_MODE_STACK_IMMD
  - UNWIND_X86_64_MODE_STACK_IND
  - UNWIND_X86_64_MODE_DWARF

- For relocatable object files the compiler can generate compact unwind information for a function by adding a row to the __LD,__compact_unwind section
- The __LD,__compact_unwind section is a table, initially with one row per each function with unwind info
- The table columns and some conceptual entries are:
  - range-start - pointer to start of function/range
  - range-length
  - compact-unwind-encoding - 32-bit encoding (above)
  - personality function or zero if no personality function is set
  - lsda or zero if no LSDA data is available.

- The __TEXT,__unwind_info section is laid out for an efficient two level lookup in final linked images
- The header of the section contains a coarse index that maps a function address to the page (4096 byte block) containing the unwind info for that function

Objects, Memory Locations, and Concurrency

```
#include <atomic>
#include <thread>
#include <string>

struct D {
    int i;
    double d;
    unsigned bf1:10;
    int bf2:25;
    int bf3:1;
    int bf4:9;
    int i2;
    char c1;
    char c2;
    std::string s;
};
std::atomic<D *> ptr{nullptr};

void producer()
{
    D *p = new D;
    p->s = "Hello";
    ptr.store(p, std::memory_order_release);
}

void consumer()
{
    D *p2 = nullptr;
    while (!(p2 = ptr.load(std::memory_order_acquire))) {
        std::this_thread::yield();
    }
}

int main()
{
    std::thread t1{producer};
    std::thread t2{consumer};
    t1.join();
    t2.join();
    return 0;
}
```

- Apple's pthread implementation was previously part of Libc
  - changed in OS X 10.9
  - library - libsystem_pthread.dylib
  - project libpthread (libpthread.dylib)
  - pthread support is in pthread.kext (kern/kern_support.c)
- libdispatch.dylib - more threading support and GCD
- see /usr/include/os (e.g. lock.h) for additional support
- sched_yield - libsystem_pthread.dylib
- switch_pri - libsystem_kernel.dylib
- atomic_load - libcompiler_rt.dylib
- mach/mach_traps.h - MACH_TRAP #59
  - /osfmk/kern/syscall_sw.c
  - /mach/syscall_sw.h

```
*** Dumping AST Record Layout
         0 | struct std::__1::__atomic_base<struct D *, false>
         0 |   _Atomic(struct D *) __a_
           | [sizeof=8, dsize=8, align=8,
           |  nvsize=8, nvalign=8]

*** Dumping AST Record Layout
         0 | struct std::__1::atomic<struct D *>
         0 |   struct std::__1::__atomic_base<struct D *, false> (base)
         0 |     _Atomic(struct D *) __a_
           | [sizeof=8, dsize=8, align=8,
           |  nvsize=8, nvalign=8]

*** Dumping AST Record Layout
         0 | struct D
         0 |   int i
         8 |   double d
    16:0-9 |   unsigned int bf1
   20:0-24 |   int bf2
    23:1-1 |   int bf3
    24:0-8 |   int bf4
        28 |   int i2
        32 |   char c1
        33 |   char c2
        40 |   class std::__1::basic_string<char> s
        40 |     class std::__1::__basic_string_common<true> (base) (empty)
...
           | [sizeof=64, dsize=64, align=8,
           |  nvsize=64, nvalign=8]
```

Objects, Memory Locations, and Concurrency

```
#include <mach/mach_traps.h>
#include <memory.h>
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <stdatomic.h>

#define NO_EXCEPTIONS
typedef pthread_t __libcpp_thread_t;

typedef struct _stdthread {
    __libcpp_thread_t __t;
} stdthread;

typedef struct _D {
    int i;
    double d;
    unsigned bf1:10;
    int bf2:25;
    int bf3:1;
    int bf4:9;
    int i2;
    char c1;
    char c2;
    char *s;

} D;

typedef _Atomic (D *) AtomicDPtr;
AtomicDPtr ptr = NULL;

typedef void (*terminate_handler)();

void default_terminate_handler()
{
    abort();
}

static terminate_handler __terminate_handler = default_terminate_handler;

terminate_handler get_terminate()
{
    return __terminate_handler;
}

int sched_yield()
{
    swtch_pri(0);
    return 0;
}

void producer()
{
    D *p = (D *) operator_new(0x40);
    DConstructor(p);
    int *__temp0 = (int *) ((void *) p);
    stdbasic_string_char_assign((char **) ((void *) p + 0x28), "Hello");
    atomic_store_explicit(&ptr, p, memory_order_release);
}

void consumer()
{
    D *p2 = 0x0;
    do {
        D *__temp1 = atomic_load_explicit(&ptr, memory_order_acquire);
        if ((((__temp1 != NULL ? 0x1 : 0x0) ^ 0xff) & 0x1) == 0x0) {
            p2 = __temp1;
            break;
        }
        sched_yield();
    } while (0x1);
}
```

Objects, Memory Locations, and Concurrency

```
static void *__thread_proxy(void *arg0)
{
    void (*__temp0)() = (void (*)()) (arg0);
    (__temp0)();
    return 0x0;
}

void __throw_system_error(int ev, const char *what_arg)
{
#ifndef NO_EXCEPTIONS
    throw system_error(error_code(ev, system_category()), what_arg);
#else
    (void) ev;
    (void) what_arg;
    abort();
#endif
}

// simplified for exposition
stdthread *stdthreadConstructorVoid(stdthread *this, void (*arg1)(void))
{
    void *__temp0 = operator_new(0x8);
    // NOTE: the constructor here is much more complex than this
    // uses __thread_proxy and __thread_struct to manage thread local data
    int __temp1 = pthread_create((pthread_t *) __temp0, 0x0, __thread_proxy,
arg1);
    int __temp2 = -24;
    if (__temp1 == 0x0) {
        *(pthread_t *) this = *(pthread_t *) __temp0;
        __temp2 = 0x0;
    } else {
        __throw_system_error(__temp1, "thread constructor failed");
    }
    if (__temp2 != 0x0) {
        if (this != 0x0) {
            operator_delete(__temp0);
        }
    }
    return this;
}

int stdthreadJoin(stdthread *arg0)
{
    int __temp0 = pthread_join(*(pthread_t *) arg0, 0x0);
    if (__temp0 == 0x0) {
        *(pthread_t *) arg0 = 0x0;
    } else {
        // allocate system_error and throw with __cxa_throw
        __cxa_throw();
    }
    return __temp0;
}

void terminate()
{
    (*get_terminate())();
    fprintf(stderr, "terminate_handler unexpectedly returned\n");
    abort();
}

stdthread *stdthreadDestructor(stdthread *arg0)
{
    if (*(pthread_t *) arg0 != 0x0) {
        terminate();
    }
    return 0x0;
}

// NOTE: thread is a wrapper for __libcpp_thread_t which is a typedef for
pthread_t on mac
int main()
{
    stdthread t1;
    stdthreadConstructorVoid(&t1, producer);
    stdthread t2;
    stdthreadConstructorVoid(&t2, consumer);
    stdthreadJoin(&t1);
    stdthreadJoin(&t2);
    stdthreadDestructor(&t1);
    stdthreadDestructor(&t2);
    return 0x0;
}
```

Benchmarks - Data Access

```
01_data_access1_no_opt
Run on (8 X 2800 MHz CPU s)
2017-07-04 09:57:01
-------------------------------------------------------------------
Benchmark                            Time           CPU Iterations
-------------------------------------------------------------------
individual_local_variables           9 ns          9 ns   82101806
local_array                          8 ns          8 ns   75693678
struct_with_public_members           8 ns          8 ns   88218985
class_inline_get                    25 ns         25 ns   28654585
class_inline_get_set                23 ns         23 ns   31080997
class_non_inline_get                20 ns         20 ns   34272565
class_non_inline_get_set            18 ns         18 ns   35467461

01_data_access1_fast_opt
Run on (8 X 2800 MHz CPU s)
2017-07-04 09:58:00
-------------------------------------------------------------------
Benchmark                            Time           CPU Iterations
-------------------------------------------------------------------
individual_local_variables           2 ns          2 ns  395348443
local_array                          2 ns          2 ns  426975065
struct_with_public_members           2 ns          2 ns  428344144
class_inline_get                     2 ns          2 ns  426766815
class_inline_get_set                 2 ns          2 ns  351898492
class_non_inline_get                 2 ns          2 ns  425113262
class_non_inline_get_set             2 ns          2 ns  349470552

01_data_access1_fast_opt
Run on (8 X 2800 MHz CPU s)
2017-07-04 09:58:31
-------------------------------------------------------------------
Benchmark                            Time           CPU Iterations
-------------------------------------------------------------------
individual_local_variables           2 ns          2 ns  411905238
local_array                          2 ns          2 ns  421661346
struct_with_public_members           2 ns          2 ns  427486137
class_inline_get                     2 ns          2 ns  429347759
class_inline_get_set                 2 ns          2 ns  345679012
class_non_inline_get                 2 ns          2 ns  427695090
class_non_inline_get_set             2 ns          2 ns  343110344


02_data_access2_no_opt
Run on (8 X 2800 MHz CPU s)
2017-07-04 09:58:58
-----------------------------------------------------------------------
Benchmark                                Time           CPU Iterations
-----------------------------------------------------------------------
single_inheritance_public_members       10 ns         10 ns   72034988
single_inheritance_inline_get_set       32 ns         32 ns   23695319
virtual_inheritance_l1_public_members   12 ns         12 ns   58154991
virtual_inheritance_l1_inline_get_set   34 ns         34 ns   20845742
virtual_inheritance_l2_public_members   13 ns         13 ns   55923497
virtual_inheritance_l2_inline_get_set   36 ns         36 ns   20459939

02_data_access2_fast_opt
Run on (8 X 2800 MHz CPU s)
2017-07-04 09:59:30
-----------------------------------------------------------------------
Benchmark                                Time           CPU Iterations
-----------------------------------------------------------------------
single_inheritance_public_members        2 ns          2 ns  372304779
single_inheritance_inline_get_set        2 ns          2 ns  343448716
virtual_inheritance_l1_public_members    4 ns          4 ns  196024072
virtual_inheritance_l1_inline_get_set    3 ns          3 ns  211695896
virtual_inheritance_l2_public_members    5 ns          5 ns  131941041
virtual_inheritance_l2_inline_get_set    5 ns          5 ns  137795276

02_data_access2_max_opt
Run on (8 X 2800 MHz CPU s)
2017-07-04 10:00:06
-----------------------------------------------------------------------
Benchmark                                Time           CPU Iterations
-----------------------------------------------------------------------
single_inheritance_public_members        3 ns          3 ns  261569792
single_inheritance_inline_get_set        2 ns          2 ns  305240988
virtual_inheritance_l1_public_members    5 ns          5 ns  142482037
virtual_inheritance_l1_inline_get_set    5 ns          5 ns  146407372
virtual_inheritance_l2_public_members    7 ns          7 ns  104056726
virtual_inheritance_l2_inline_get_set    7 ns          7 ns  105772137
```

Benchmarks

Data Member and Pointer to Data Member Access

```
03_nonstatic_data_member_access_no_opt
Run on (8 X 2800 MHz CPU s)
2017-07-04 10:00:43
-------------------------------------------------------------
Benchmark                      Time           CPU Iterations
-------------------------------------------------------------
direct_access                   8 ns          8 ns   80734453
bound_access                    9 ns          9 ns   68693451
pointer_to_data_member         10 ns         10 ns   73459193
```

```
03_nonstatic_data_member_access_fast_opt
Run on (8 X 2800 MHz CPU s)
2017-07-04 10:01:04
-------------------------------------------------------------
Benchmark                      Time           CPU Iterations
-------------------------------------------------------------
direct_access                   2 ns          2 ns  399611806
bound_access                    2 ns          2 ns  427985351
pointer_to_data_member          2 ns          2 ns  418370030
```

```
03_nonstatic_data_member_access_max_opt
Run on (8 X 2800 MHz CPU s)
2017-07-04 10:01:40
-------------------------------------------------------------
Benchmark                      Time           CPU Iterations
-------------------------------------------------------------
direct_access                   3 ns          3 ns  266019100
bound_access                    2 ns          2 ns  280386934
pointer_to_data_member          2 ns          2 ns  288341132
```

```
04_pointer_to_data_member_access_no_opt
Run on (8 X 2800 MHz CPU s)
2017-07-04 10:02:10
-------------------------------------------------------------------------------
Benchmark                                        Time           CPU Iterations
-------------------------------------------------------------------------------
pointer_to_data_member_no_inheritance            9 ns          9 ns   69609491
pointer_to_data_member_single_inheritance       10 ns         10 ns   71085475
pointer_to_data_member_vi1                      12 ns         12 ns   59504076
pointer_to_data_member_vi2                      12 ns         12 ns   56552861
```

```
04_pointer_to_data_member_access_fast_opt
Run on (8 X 2800 MHz CPU s)
2017-07-04 10:02:34
-------------------------------------------------------------------------------
Benchmark                                        Time           CPU Iterations
-------------------------------------------------------------------------------
pointer_to_data_member_no_inheritance            2 ns          2 ns  381425660
pointer_to_data_member_single_inheritance        2 ns          2 ns  425237222
pointer_to_data_member_vi1                       4 ns          4 ns  184234768
pointer_to_data_member_vi2                       6 ns          6 ns  129160824
```

```
04_pointer_to_data_member_access_max_opt
Run on (8 X 2800 MHz CPU s)
2017-07-04 10:02:54
-------------------------------------------------------------------------------
Benchmark                                        Time           CPU Iterations
-------------------------------------------------------------------------------
pointer_to_data_member_no_inheritance            3 ns          3 ns  261726272
pointer_to_data_member_single_inheritance        3 ns          3 ns  248820767
pointer_to_data_member_vi1                       5 ns          5 ns  143423226
pointer_to_data_member_vi2                       7 ns          7 ns  100201835
```

Benchmarks

Function Performance

```
05_function_performance_no_opt
Run on (8 X 2800 MHz CPU s)
2017-07-04 10:03:17
-----------------------------------------------------------
Benchmark                    Time           CPU Iterations
-----------------------------------------------------------
inline_member               12 ns         12 ns   53317897
nonmember_function          12 ns         12 ns   58901735
static_member               11 ns         11 ns   54046959
nonstatic_member            12 ns         12 ns   60348990
virtual_member              14 ns         14 ns   55480701

05_function_performance_fast_opt
Run on (8 X 2800 MHz CPU s)
2017-07-04 10:03:44
-----------------------------------------------------------
Benchmark                    Time           CPU Iterations
-----------------------------------------------------------
inline_member                2 ns          2 ns  329278504
nonmember_function           2 ns          2 ns  347084490
static_member                2 ns          2 ns  302481646
nonstatic_member             2 ns          2 ns  339916381
virtual_member               2 ns          2 ns  298273423

05_function_performance_max_opt
Run on (8 X 2800 MHz CPU s)
2017-07-04 10:04:05
-----------------------------------------------------------
Benchmark                    Time           CPU Iterations
-----------------------------------------------------------
inline_member                2 ns          2 ns  342288246
nonmember_function           2 ns          2 ns  310866563
static_member                2 ns          2 ns  301903719
nonstatic_member             2 ns          2 ns  304440482
virtual_member               2 ns          2 ns  338848496
```

Benchmarks

Memberwise Initialization, Copies

```
06_memberwise_initialization_and_copy_no_opt
Run on (8 X 2800 MHz CPU s)
2017-07-04 10:04:42
------------------------------------------------------------------------------------------
Benchmark                                                   Time           CPU Iterations
------------------------------------------------------------------------------------------
struct_member_init                                        10 ns         10 ns   73326839
struct_copies                                             13 ns         13 ns   45260864
class_public_member_init                                  10 ns         10 ns   70883205
class_public_copies                                       13 ns         13 ns   55280469
class_private_member_init_inline                          16 ns         16 ns   43724312
class_private_member_init_ctor                            13 ns         13 ns   53282588
class_private_copies                                      14 ns         14 ns   44990038
class_private_member_non_inline                           16 ns         16 ns   44535211
class_single_inheritance_inline                           17 ns         17 ns   41954605
class_single_inheritance_copies                           15 ns         15 ns   46053844
class_multiple_inheritance_inline                         19 ns         19 ns   36576063
class_multiple_inheritance_copies                         15 ns         15 ns   46432338
class_single_inheritance_synthesized                      17 ns         17 ns   38892778
class_single_inheritance_synthesized_copies               13 ns         13 ns   54081617
class_multiple_inheritance_synthesized                    42 ns         42 ns   15941336
class_multiple_inheritance_synthesized_copies             74 ns         74 ns    9255219
class_virtual_inheritance_2_levels_synthesized            49 ns         49 ns   13931180
class_virtual_inheritance_2_levels_synthesized_copies     95 ns         95 ns    7586842


06_memberwise_initialization_and_copy_fast_opt
Run on (8 X 2800 MHz CPU s)
2017-07-04 10:05:17
------------------------------------------------------------------------------------------
Benchmark                                                   Time           CPU Iterations
------------------------------------------------------------------------------------------
struct_member_init                                         2 ns          2 ns  324717496
struct_copies                                              4 ns          4 ns  195485404
class_public_member_init                                   2 ns          2 ns  329714326
class_public_copies                                        4 ns          4 ns  191992803
class_private_member_init_inline                           2 ns          2 ns  342821322
class_private_member_init_ctor                             2 ns          2 ns  346133687
class_private_copies                                       3 ns          3 ns  257614344
class_private_member_non_inline                            2 ns          2 ns  343290389
class_single_inheritance_inline                            2 ns          2 ns  340263364
class_single_inheritance_copies                            3 ns          3 ns  255840473
class_multiple_inheritance_inline                          2 ns          2 ns  306782075
class_multiple_inheritance_copies                          3 ns          3 ns  241332708
class_single_inheritance_synthesized                       2 ns          2 ns  310092629
class_single_inheritance_synthesized_copies                3 ns          3 ns  279342985
class_multiple_inheritance_synthesized                     2 ns          2 ns  320733104
class_multiple_inheritance_synthesized_copies              4 ns          4 ns  160169139
class_virtual_inheritance_2_levels_synthesized             2 ns          2 ns  339899875
class_virtual_inheritance_2_levels_synthesized_copies      7 ns          7 ns   96246391


06_memberwise_initialization_and_copy_max_opt
Run on (8 X 2800 MHz CPU s)
2017-07-04 10:05:57
------------------------------------------------------------------------------------------
Benchmark                                                   Time           CPU Iterations
------------------------------------------------------------------------------------------
struct_member_init                                         2 ns          2 ns  336220022
struct_copies                                              4 ns          4 ns  200010858
class_public_member_init                                   2 ns          2 ns  333652687
class_public_copies                                        4 ns          4 ns  194681307
class_private_member_init_inline                           2 ns          2 ns  329804426
class_private_member_init_ctor                             2 ns          2 ns  350722488
class_private_copies                                       3 ns          3 ns  270820276
class_private_member_non_inline                            2 ns          2 ns  346859190
class_single_inheritance_inline                            2 ns          2 ns  352171135
class_single_inheritance_copies                            3 ns          3 ns  267979511
class_multiple_inheritance_inline                          2 ns          2 ns  352064860
class_multiple_inheritance_copies                          3 ns          3 ns  275736018
class_single_inheritance_synthesized                       2 ns          2 ns  339759644
class_single_inheritance_synthesized_copies                3 ns          3 ns  276009999
class_multiple_inheritance_synthesized                     2 ns          2 ns  330581635
class_multiple_inheritance_synthesized_copies              4 ns          4 ns  163081970
class_virtual_inheritance_2_levels_synthesized             2 ns          2 ns  327453209
class_virtual_inheritance_2_levels_synthesized_copies      7 ns          7 ns   98024114
```

Afterthoughts

- Many language features are completely or partially handled by the frontend
  - constness
    - clang/lib/Sema - ConstQualifier, ConstantExpression, Constexpr
    - backend - llvm/include/llvm/IR/Constants.h
  - move semantics and forwarding
- Howard Hinnant on recent cppcast
  - move semantics does not have complex underlying machinery
  - it is compile-time magic, i.e. a compile time switch to guide overload resolution
- It can be difficult to sort out the source of a feature
  - libcxx
  - libcxxabi
  - compiler_rt
  - system libraries (libSystem.B.dylib, libsystem_pthread.dylib, etc.)
  - open source vs. closed source

Conclusions

- C separates data and the operations on that data and was designed as a portable assembly (it still mirrors that design relatively closely)
- C++ combines data and the operations on that data but this requires the addition of runtime machinery to handle methods, polymorphism, generics, RTTI, and exceptions
  - These describe the essence of the object model and the steps towards higher-level abstractions
  - However, C++ strives to maintain “zero overhead abstraction” and compatibility (for the most part) with C
- This machinery results in a complex network of compilers, compiler libraries, compiler runtimes, OS and related runtime libraries, the language runtime, and the C++ ABI
- Because of the nature of C++, a main goal is often to limit overhead (memory/CPU)
- It can be daunting to sort language features and the resulting overhead out
  - tools exist to help a curious mind remove the guess work
  - Use these tools as much as possible

References

- Inside the C++ Object Model - Stanley B. Lippman
- Intel 64 and IA-32 Architectures Software Developer's Manual
- Modern X86 Assembly Language Programming Daniel Kusswurm
- https://itanium-cxx-abi.github.io/cxx-abi/abi.html
- https://libcxxabi.llvm.org/
- http://en.cppreference.com/w/ - various articles and examples
- http://llvm.org/docs/doxygen
- Getting Started with LLVM Core Libraries - Bruno Cardoso Lopes, Rafael Auler
- https://en.wikipedia.org/wiki/Macintosh_operating_systems
- https://en.wikipedia.org/wiki/Mach_(kernel)
- https://en.wikipedia.org/wiki/Mach-O
- http://math-atlas.sourceforge.net/devel/assembly/MachORuntime.pdf
- https://developer.apple.com/library/content/documentation/DeveloperTools/Conceptual/MachOTopics/1-Articles/loading_code.html
- https://www.objc.io/issues/6-build-tools/mach-o-executables/
- http://www.objs.com/x3h7/fmindex.htm
- Modern Compiler Design - Dick Grune
- man dyld
- n4659 draft of the ISO International Standard ISO/IEC – Programming Language C++
- https://en.wikipedia.org/wiki/Runtime_system
- https://en.wikipedia.org/wiki/X86_memory_segmentation
- http://wiki.osdev.org/Stack_Smashing_Protector
- https://stackoverflow.com/questions/3149175/what-is-the-difference-between-trap-and-interrupt
- http://newosxbook.com/articles/GCD.html
- other sources cited in the related slides