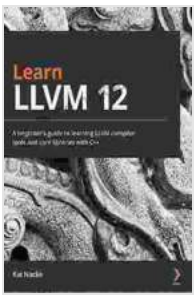
Beginner Guide to Learning LLVM Compiler Tools and Core Libraries with Comprehensive Examples

The LLVM Compiler Infrastructure is a collection of modular and reusable compiler and toolchain technologies. It provides a common infrastructure for building compiler front ends, optimizers, and code generators for various programming languages and architectures. LLVM is used in a wide range of projects, including the Clang compiler, the Swift compiler, and the Rust compiler.

This guide is designed to help you get started with LLVM. We will cover the basics of LLVM, including its architecture, its core components, and how to use it to build your own compiler tools. We will also provide a number of examples that demonstrate how to use LLVM to solve real-world problems.

LLVM is a modular compiler infrastructure. This means that it is composed of a number of independent components that can be assembled in different ways to create different compilers. The core components of LLVM are:



Learn LLVM 12: A beginner's guide to learning LLVM compiler tools and core libraries with C++ by Kai Nacke

4.2 out of 5

Language	: English
File size	: 2442 KB
Text-to-Speech	: Enabled
Enhanced typesetting	: Enabled
Print length	: 392 pages
Screen Reader	: Supported

FREE DOWNLOAD E-BOOK

- **The LLVM Intermediate Representation (IR)**: The IR is a low-level representation of code that is independent of any particular programming language or architecture. It is used to represent the output of the front end and the input to the back end.
- **The LLVM Optimizer:** The optimizer is a collection of passes that can be used to improve the performance of code. The optimizer can perform a variety of optimizations, including common subexpression elimination, constant propagation, and loop unrolling.
- **The LLVM Code Generator:** The code generator is a component that translates the IR into machine code. The code generator can target a variety of architectures, including x86, ARM, and MIPS.

In addition to these core components, LLVM also provides a number of other tools, including a debugger, a profiler, and a disassembler. These tools can be used to help you develop and debug your compiler tools.

The best way to get started with LLVM is to download the LLVM source code and build it. The LLVM source code can be found at:

https://llvm.org/releases/download.html

Once you have downloaded the LLVM source code, you can build it by following the instructions in the LLVM documentation.

Once you have built LLVM, you can start using it to build your own compiler tools. To do this, you will need to create a new LLVM project. A new LLVM

project can be created by running the following command:

llvm-project new my-project

This command will create a new directory called `my-project`. The `my-project` directory will contain a number of files, including a `CMakeLists.txt` file and a `main.cpp` file.

The `CMakeLists.txt` file is used to configure the build process for your project. The `main.cpp` file is the entry point for your project.

To build your project, you will need to run the following command:

cmake . && make

This command will build your project and create an executable file called `my-project`.

You can now run your project by running the following command:

./my-project

This command will run your project and print the following message:

Hello, world!

One of the most common ways to use LLVM is to create a new LLVM pass. A pass is a function that is called on each function in a module. Passes can be used to perform a variety of tasks, such as optimization, code generation, and debugging.

To create a new LLVM pass, you will need to create a new class that inherits from the `llvm::Pass` class. The following code shows how to create a new pass that prints the name of each function in a module:

cpp #include "llvm/Pass.h"

using namespace llvm;

namespace { struct MyPass : public Pass { static char ID;

```
  MyPass() : Pass(ID) {}bool runOnModule(Module &amp;M) override { fo
```

To use your new pass, you will need to add it to the pass manager. The following code shows how to add your pass to the pass manager:

cpp PassManager PM; PM.add(new MyPass()); PM.run(M);

This code will add your pass to the pass manager and run it on the module `M`.

In this section, we will provide a number of examples that demonstrate how to use LLVM to solve real-world problems.

Example 1: Optimizing a Function

The following example shows how to use LLVM to optimize a function. The function we will optimize is a simple function that computes the factorial of a number.

cpp int factorial(int n){if (n == 0){return 1; }else { return n * factorial(n - 1); }}

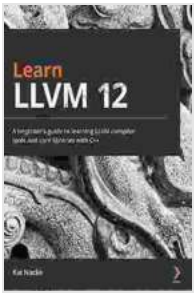
We can use LLVM to optimize this function by using the `-O2` optimization level. The following code shows how to optimize the function using LLVM:

cpp #include "llvm/IR/Module.h" #include "llvm/IR/Function.h" #include "llvm/IR/BasicBlock.h" #include "llvm/IR/Instruction.h" #include "llvm/Passes/PassManager.h" #include "llvm/Transforms/Scalar.h" #include "llvm/Support/TargetSelect.h"

using namespace llvm;

int main(){LLVMInitializeAllTargetInfos(); LLVMInitializeAllTargetMCs(); LLVMInitializeAllAsmParsers();

```
Module M("my-module"); Function *F = Function::Create( FunctionType:
```

Learn LLVM 12: A beginner's guide to learning LLVM compiler tools and core libraries with C++ by Kai Nacke

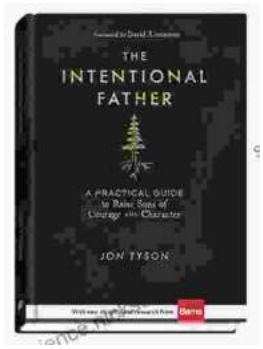
4.2 out of 5

Language	: English
File size	: 2442 KB
Text-to-Speech	: Enabled
Enhanced typesetting	: Enabled
Print length	: 392 pages
Screen Reader	: Supported

FREE DOWNLOAD E-BOOK

Compilation of Short Stories on Mental Illness and Ways to Handle Them

Mental illness is a serious issue that affects millions of people around the world. It can be a debilitating condition that can make it difficult to live a normal life....



The Practical Guide to Raising Courageous and Characterful Sons

As parents, we all want our sons to grow up to be good men. We want them to be kind, compassionate, and brave. We want them to stand up for what they...