https://ForgeVM.org

مسودة

هندسة لغة برمجة باستخدام LLVM

من الفكرة إلى مترجم جاهز للإنتاج

إعداد : أيمن الحراكي

هندسة لغة برمجة باستخدام LLVM

من الفكرة إلى مترجم جاهز للإنتاج

تم دعم بعض أعمال الصياغة واستكشاف الأفكار باستخدام أدوات ذكاء اصطناعي حديثة،

وذلك تحت إشراف كامل، مع التحقق والمراجعة والتصحيح، وبمسؤولية وتأليف كاملين.

إعداد : أيمن الحراكي

ForgeVM.org

مارس 2026

المحتويات

الخلاصة

استعراض خط الأنابيب الكامل

تناول هذا الكتاب المترجم المبني على LLVM بوصفه نظامًا إنتاجيًا متكاملاً، وليس مجرد برنامج تنفيذي واحد «يحوّل الشيفرة المصدرية إلى شيفرة آلة». إن خط أنابيب المترجم الحديث هو سلسلة من العقود الهندسية: كل مرحلة تستقبل أثرًا برمجيًا (artifact) ذو ثوابت محددة بدقة، تقوم بتحويله، ثم تُصدر أثرًا جديدًا بثوابت أقوى، أو أدنى مستوى، أو أكثر تخصصًا. وعلى نظام Windows يجب أن تراعي هذه السلسلة كذلك واقع اكتشاف الأدوات، ومعالجة المسارات، وقواعد الاقتباس، وملفات الاستجابة، والتكامل مع أنظمة البناء.

تصميم اللغة

اللغة ليست مجرد صياغة نحوية. التعريف الإنتاجي للغة هو مجموعة القواعد التي تجعل الترجمة والأدوات المصاحبة لها قابلة للتنبؤ:

- النموذج المعجمي: فئات الرموز، قواعد الأعداد، قواعد الهروب في السلاسل النصية، وأشكال التعليقات.
- القواعد النحوية: تراكيب قابلة للتحليل مع أسبقية وترابط غير ملتبسين، واستراتيجية تعافٍ من الأخطاء للتشخيص.

- قواعد الأسماء والنطاق: الوحدات/المجالات الاسمية، الرؤية، قواعد الاستيراد، ودلالات حل الرموز.
- نظام الأنواع: الأنواع البدائية والمركبة، الأنماط العامة (إن وجدت)، التحويلات، وحدود الاستدلال النوعي.
- نموذج الذاكرة والكائن: الأعمار، القابلية للتغيير، التداخل، سياسة الملكية (إن وجدت)، وافتراضات التوازي.
- سطح **ABI**: اتفاقيات النداء، قواعد التخطيط، واستراتيجية التكامل مع C وواجهات نظام التشغيل.

الدرس الإنتاجي الأساسي هو أن كل «ميزة جميلة» في اللغة يجب أن يُدعَم بـ:

- قاعدة دلالية ساكنة واضحة،
- وثابت يمكن التحقق منه حتميًا داخل المترجم،
- واختيار تمثيل مستقر في IR.

بناء الواجهة الأمامية

الواجهة الأمامية هي «مترجم داخل المترجم». إنها تحوّل النص إلى معنى:

- المحلل المعجمي: تقسيم سريع وحتمي مع مواقع مصدر صحيحة.
- المحلل النحوي: بناء شجرة البنية مع تعافٍ من الأخطاء يحافظ على أكبر قدر من الهيكل.
- التحليل الدلالي: ربط الأسماء، فحص الأنواع، حل التحميل الزائد، وتقييم الثوابت.
- التشخيص: أخطاء وتحذيرات وملاحظات موجهة للمستخدم بصياغة مستقرة ونطاقات دقيقة.

الواجهة الأمامية الإنتاجية توفر:

- حدًا فاصلاً مستقرًا بين AST وIR المطبّع،
- معالجة أخطاء حتمية لا تُخفي السبب الجذري،
- وتصميمًا يدعم إضافة ميزات تدريجية دون إعادة كتابة شاملة.

توليد IR

توليد IR هو المرحلة التي يتحول فيها عقد اللغة إلى عقد LLVM. المسؤولية الجوهرية هي الحفاظ على الدلالات أثناء الإسقاط إلى نموذج LLVM:

- صحة الوحدة: يجب أن يتطابق target triple وdata layout مع الهدف المقصود.
- صحة الدوال: يجب أن تتطابق اتفاقية النداء والخصائص والربط مع ABI ونوايا التحسين.
- صحة الذاكرة: يجب أن تكون فضاءات العناوين والمحاذاة وقواعد التداخل صريحة.
- صحة معلومات التصحيح: يجب إصدار مواقع المصدر وتتبع المتغيرات بشكل منهجي.

الانضباط المركزي: لا تُشغّل تحسينات واعية بالهدف على وحدة قبل ضبط *triple* و*datalayout* الصحيحين.

التحسين

التحسين ليس مفتاحًا واحدًا، بل مجموعة مراحل تُختار بسياسة واضحة:

- مستوى **IR**: التطبيع، التبسيط، تحسينات القيم العددية، تحويلات الحلقات، وفرص المتجهة.
- مستوى التوليد الخلفي: اختيار التعليمات، الجدولة، تخصيص السجلات، ترتيب الكتل، وضبط خاص بالهدف.

السلسلة الإنتاجية تتعامل مع التحسين كسطح قابل للتحكم:

- سياسات O- مستقرة،
- دعم اختياري لـ LTO/PGO,
- وآليات إعادة إنتاج موثوقة (###-، حفظ الوسائط، سلاسل حتمية).

إنتاج الملفات الكائنية

إنتاج الملف الكائني هو الانتقال من IR إلى أثر ثنائي خاص بالمنصة:

- اختيار تنسيق الملف: COFF لأهداف Windows، وMach-O/ELF للعبور.
- الترحيلات والمقاطع: وضع المقاطع والمحاذاة وإصدار الترحيلات بشكل صحيح.
- مراجع زمن التنفيذ: يجب أن تُحل الدوال المساعدة وواجهات ABI أثناء الربط.

على Windows يجب التحقق من نجاح إنتاج الملف الكائني عبر:

- جداول الرموز،
- جداول الترحيل،
- وقدرة الرابط على استهلاكها وحلها.

الربط

الربط هو المرحلة التي يتحول فيها المترجم إلى سلسلة أدوات كاملة:

- نوع الرابط: نمط COFF (مثل lld-link) لبيئات شبيهة بـ MSVC.
- نموذج البحث: sysroot، مسارات المكتبات، اكتشاف كائنات الإقلاع.

- أنواع الأثر: تنفيذيات، مكتبات ديناميكية، مكتبات ساكنة، أو صور مدمجة مستقلة.

يجب أن يكون الربط قابلاً لإعادة الإنتاج:

- طباعة أمر الربط،
- دعم ملفات الاستجابة،
- إعدادات افتراضية حتمية تعتمد عليها أنظمة البناء.

مرجع خط أنابيب متكامل على Windows

الأوامر التالية تلخص خطًا عمليًا من البداية إلى النهاية. استبدل mycc.exe بسائقك الخاص.

(1) من الواجهة الأمامية إلى LLVM IR

```
mycc.exe -emit-llvm -S src\main.mylang -o out\main.ll
```

(2) تحسينات المستوى المتوسط

```
opt.exe -S -O2 out\main.ll -o out\main.opt.ll
```

(3) توليد ملف كائني

```
llc.exe -filetype=obj out\main.opt.ll -o out\main.obj
```

(4) الربط إلى تنفيذية

```
lld-link.exe out\main.obj /OUT:out\app.exe
```

(5) إعادة الإنتاج والتصحيح

```
mycc.exe -### -O2 src\main.mylang -o out\app.exe
mycc.exe -save-temps -O2 src\main.mylang -o out\app.exe
```

(6) الترجمة العابرة

```
mycc.exe -target aarch64-unknown-linux-gnu ^
  --sysroot=C:\Toolchains\aarch64-linux-gnu\sysroot ^
  -O2 -c src\main.mylang -o out\main.o
```

منجزات الدليل

بنهاية هذا الدليل أصبحت تملك مخططًا لبناء سلسلة أدوات كاملة مبنية على LLVM:

- استراتيجية متماسكة لإسقاط اللغة إلى IR.
- معمارية سائق مترجم تحول argv إلى رسم بياني للأفعال.
- نموذج عملي للترجمة العابرة باستخدام triples وsysroots.
- رؤية متكاملة لهندسة زمن التنفيذ.
- منهجية اختبار مستوحاة من ممارسات LLVM.

- سير عمل مضبوط لهندسة الأداء (LTO/PGO).
- مقدمة إلى MLIR وفلسفة multi-level IR.
- دليل رفيع المستوى لتوسيع الواجهة الخلفية.

الأهم أنك تمتلك الآن منهجًا هندسيًا قابلاً للتكرار:

- عرّف الثوابت،
- شفّرها داخل المترجم،
- اختبرها كعقود،
- وطوّر دون كسر المستخدمين الإنتاجيين.

مسارات التطوير المستقبلية

المترجم الإنتاجي لا ينتهي؛ بل يستقر ثم يتطور:

توسعة اللغة والأدوات

- نظام وحدات ونموذج حزم.
- ترجمة تدريجية.
- خادم لغة.
- منسّق ولِنتر مبنيان على نفس AST.

خارطة طريق الأداء

- تحسينات واعية باللغة قبل الخفض.
- PGO على مستوى أحمال العمل.
- ThinLTO.
- سياسات تقليل الحجم.

النمو المنصاتي وABI

- أوضاع سائق متعددة على Windows.
- أهداف عابرة مُختبرة ومعبأة.
- سياسة واضحة لتنسيقات التصحيح.

MLIR ومسارات DSL

- تبني MLIR للأنظمة الغنية دلاليًا.
- خطوط خفض متعددة المستويات.
- عزل لهجات المجال عن لهجات الخفض.

الانتقال من نموذج أولي إلى مترجم صناعي

الانتقال ليس إعادة كتابة، بل تقوية منهجية للعقود والاختبارات والانضباط الهندسي.

1) تثبيت الواجهات والثوابت

- سلوك CLI مستقر.
- تنسيق تشخيص ثابت.
- قواعد إصدار IR مستقرة.
- سلوك ربط حتمي على Windows.

2) تحويل الأخطاء إلى اختبارات

```
mycc.exe -emit-llvm -S %s -o - | FileCheck.exe %s
```

3) جعل التغليف جزءاً من المنتج

- اكتشاف الأدوات نسبيًا إلى السائق.
- دليل موارد حتمي.
- دعم ملفات الاستجابة.

4) وضع ميزانيات أداء وصحة

```
mycc.exe -O2 -fprofile-instr-generate src\app.mylang -o out\app-instr.exe
set LLVM_PROFILE_FILE=out\app.profraw
out\app-instr.exe
llvm-profdata.exe merge -output=out\app.profdata out\app.profraw
mycc.exe -O2 -fprofile-instr-use=out\app.profdata src\app.mylang -o
↪  out\app-pgo.exe
```

5) تشخيص صناعي الجودة

- أوضاع تفصيلية.
- إخراج إعادة إنتاج.
- بوابات تحقق داخلية.

6) حوكمة وإصدار

- سياسة إصدار.
- سياسة توافق ABI/IR.
- إصدارات بدورات واضحة.

منظور أخير

LLVM يوفر البنية التحتية، لا المنتج. المنتج هو لغتك ودلالاتها وموثوقية سلسلة أدواتك. الانتقال الناجح إلى مترجم صناعي يتطلب انضباطًا هندسيًا:

- الحفاظ على المعنى عبر مراحل الخفض،
- التحقق المبكر والمستمر من الثوابت،
- اختبار العقود عند كل حد،
- وشحن سلسلة أدوات تتصرف بالطريقة نفسها على كل جهاز Windows.

ما بنيته ليس مجرد خط أنابيب ترجمة، بل منصة إنتاجية لنظام بيئي لغوي كامل.

الملاحق

الملحق A — خريطة الوثائق الرسمية لمشروع LLVM

يوفّر هذا الملحق خريطة منظمة للتسلسل الهرمي للوثائق الرسمية لمشروع LLVM كما هو محفوظ في شجرة المصدر الرسمية وكما يُنشر في بوابة التوثيق الرسمية. الهدف هو توجيه مهندس المترجمات الإنتاجية إلى المراجع الأولية المعتمدة بدل الاعتماد على ملخصات ثانوية.

المعمارية الأساسية لـ LLVM

تشمل مناطق التوثيق المفاهيمي الأساسية ما يلي:

- دليل مرجع لغة **LLVM (LLVM Language Reference Manual)** المرجع النهائي لصياغة ودلالات LLVM IR. يغطي:
  - نظام الأنواع
  - التعليمات
  - السمات (Attributes)
  - اتفاقيات النداء
  - البيانات الوصفية ومعلومات التصحيح

- نموذج الذاكرة والدلالات الذرية

- **(Programmer's Manual) المبرمج دليل** أنماط استخدام واجهات C++ API:
  - استخدام `IRBuilder`
  - إدارة الوحدات والسياقات
  - بنية التمريرات (Pass Infrastructure)
  - أدوات التحليل

- توثيق التمريرات (Passes Documentation) نظرة عامة على تمريرات التحسين وبنية سلاسل التمرير.

- توثيق مولد الشيفرة **(Code Generator Documentation)** معمارية الواجهة الخلفية:
  - SelectionDAG
  - GlobalISel
  - تخصيص السجلات
  - Machine IR

توثيق الأهداف والواجهة الخلفية

الوثائق الأساسية لتطوير الواجهات الخلفية:

- Writing an LLVM Backend
- TableGen Reference
- CodeGen Internals
- MC Layer Documentation

توثيق MLIR

المراجع الخاصة بـ MLIR تشمل:

- MLIR Language Reference
- تعريف اللهجات و ODS
- إعادة كتابة الأنماط والتحويل
- بنية التمريرات

توثيق الأدوات

أدوات سطر الأوامر:

- clang
- opt
- llc
- llvm-link
- llvm-ar
- llvm-objdump
- llvm-readobj
- llvm-profdata
- llvm-cov

على نظام Windows تُثبَّت هذه الأدوات عادة بصيغة exe.. للتحقق من توفرها:

```
where llc
where opt
where llvm-profdata
```

الملحق B — الهيكل الكامل لمشروع مترجم

عادةً ما يتبع مشروع مترجم إنتاجي مبني على LLVM البنية التالية:

```
MyCompiler/
  CMakeLists.txt
  cmake/
  include/
    mylang/
      AST/
      Semantic/
      CodeGen/
      Driver/
  lib/
    AST/
    Semantic/
    CodeGen/
    Driver/
  tools/
    mycc/
  runtime/
    builtins/
    startup/
  tests/
```

```
    unit/
    ir/
    regression/
```

هيكل السائق (مدرك لـ Windows)

```
int main(int argc, char** argv) {
  Driver driver;
  Compilation compilation = driver.buildCompilation(argc, argv);
  return driver.execute(compilation);
}
```

مثال تكامل CMake

```
find_package(LLVM REQUIRED CONFIG)

add_executable(mycc
  tools/mycc/main.cpp
)

target_link_libraries(mycc
  PRIVATE
  LLVMCore
  LLVMIRReader
  LLVMBitWriter
  LLVMCodeGen
  LLVMSupport
)
```

الملحق C — مرجع ABI للمنصات الرئيسية

يجب أن يلتزم المترجم بعقود ABI. فيما يلي ملخص عالي المستوى:

Windows x64 ABI (COFF)

- أول أربعة معاملات صحيحة: RCX, RDX, R8, R9
- محاذاة المكدس 16 بايت قبل الاستدعاء
- مساحة ظل (32 بايت) يحجزها المستدعي
- سجلات محفوظة عبر الاستدعاء: RBX, RBP, RDI, RSI, R12–R15

System V AMD64 ABI (ELF)

- معاملات صحيحة: RDI, RSI, RDX, RCX, R8, R9
- معاملات عائمة: XMM0–XMM7
- منطقة حمراء 128 بايت

AArch64 ABI (AAPCS64)

- معاملات صحيحة: X0–X7
- معاملات عائمة: V0–V7
- محاذاة مكدس 16 بايت

يتطلب التنفيذ الصحيح لـ ABI:

- سلسلة DataLayout صحيحة
- اختيار اتفاقية النداء المناسبة
- خفضًا صحيحًا للتراكيب والمتغيرات المتغيرة المعاملات

الملحق D — مرجع تنسيقات الملفات الكائنية والتنفيذية

COFF (Windows)

- رؤوس المقاطع
- جدول الرموز
- إدخالات الترحيل
- جداول الاستيراد/التصدير (في طبقة PE)

الفحص على Windows:

```
llvm-readobj.exe --file-headers file.obj
llvm-objdump.exe -d file.obj
```

ELF

- رأس ELF
- رؤوس البرامج
- رؤوس المقاطع
- جداول الرموز
- الترحيلات (REL/RELA)

Mach-O

- أوامر التحميل
- أوامر المقاطع
- LC_SYMTAB

الملحق ٤ — ملخص تعليمات LLVM IR

الحسابية

```
%r = add i32 %a, %b
%r = sub i32 %a, %b
%r = mul i32 %a, %b
```

الذاكرة

```
%p = alloca i32
store i32 %v, ptr %p
%v = load i32, ptr %p
```

تدفق التحكم

```
br label %target
br i1 %cond, label %iftrue, label %iffalse
```

الدوال

```
define i32 @f(i32 %a) {
entry:
  ret i32 %a
}
```

الملحق F — أخطاء شائعة في LLVM

- نسيان ضبط target triple قبل التحسين.
- عدم تطابق سلسلة DataLayout.
- افتراضات محاذاة غير صحيحة.
- إصدار اتفاقية نداء خاطئة.
- عدم التحقق من الوحدات.

التحقق دائمًا:

```
opt.exe -verify input.ll
```

الملحق G — ملاحظات الانتقال بين إصدارات LLVM

يتطور LLVM باستمرار. أنماط الانتقال الشائعة:

- الانتقال من مدير التمريرات القديم إلى الجديد.
- اعتماد نموذج المؤشرات المعتمة.

- إعادة تسمية وإعادة هيكلة واجهات CodeGen.
- تطور GlobalISel.
- توسع واجهات MLIR.

أفضل الممارسات:

- تثبيت نسخة LLVM في CI.
- البناء مع تمكين Assertions.
- مراجعة ملاحظات الإصدار لكل تحديث.
- الحفاظ على طبقة توافق داخل حدود التجريد الخاصة بك.

مثال فحص نسخة على Windows:

```
llvm-config.exe --version
```

تذكير هندسي أخير

يوفر LLVM بنية تحتية قوية ومتطورة. هندسة مترجم صناعي تتطلب:

- عقود دلالية واضحة
- سلوك بناء وربط حتمي
- التزامًا دقيقًا بـ ABI
- تحققًا من ثوابت IR
- اختبار انحداري مستمر

المترجم الإنتاجي لا يُعرّف فقط بقدرته على توليد الشيفرة، بل باستقراره، وقابليته لإعادة الإنتاج، وانضباط صيانته طويلة الأمد.

المراجع

وثائق LLVM الرسمية

١. **LLVM Project**. *LLVM Language Reference Manual (LangRef)*.

٢. **LLVM Project**. *LLVM Programmer's Manual*.

٣. **LLVM Project**. *Using the New Pass Manager*.

٤. **LLVM Project**. *Writing an LLVM Pass (New Pass Manager)*.

٥. **LLVM Project**. *ORC JIT: Design and Implementation (ORCv2)*.

٦. **LLVM Project**. *The LLVM Target-Independent Code Generator*.

٧. **LLVM Project**. *Writing an LLVM Backend*.

٨. **LLVM Project**. *TableGen Programmer's Reference*.

٩. **LLVM Project**. *Machine IR (MIR) Format Reference Manual*.

مراجع ABI

١. System V AMD64 ABI Supplement.

٢. Microsoft x64 Calling Convention Documentation.

٣. Arm AAPCS64 Procedure Call Standard.

مواصفات تنسيقات التنفيذ والكائن

١. ELF Specification (TIS / gABI).

٢. PE/COFF Specification (Microsoft).

٣. Mach-O Runtime Architecture Documentation.

الأسس الأكاديمية

١. Cytron et al., Static Single Assignment Form (TOPLAS, 1991).

٢. Aho et al., Compilers: Principles, Techniques, and Tools.

٣. Muchnick, Advanced Compiler Design and Implementation.

٤. Cooper and Torczon, Engineering a Compiler.

سياسة استخدام المراجع في هذا الكتاب

- دلالات LLVM مستندة أولاً إلى وثائقه الرسمية.

- قواعد ABI مستندة إلى المواصفات الرسمية للمنصات.
- تنسيقات الملفات مستندة إلى مواصفاتها القياسية.
- المراجع الأكاديمية تبرر المنهجية وتشرح الخلفية النظرية.