Hello World from Scratch

Peter Bindels
he/him
@dascandy42
Principal Software Engineer
TomTom

Simon Brand
they/them
@tartanllama
C++ Developer Advocate
Microsoft
Hello World from Scratch

Peter Bindels
he/him
@dascandy42
Principal Software Engineer
TomTom

Simon Brand
they/them
@tartanllama
C++ Developer Advocate
Microsoft
“If you wish to make an apple pie from scratch, you must first invent the universe.”

— Carl Sagan, *Cosmos*
This phrase: “making an apple pie from scratch” has a really deep meaning if you look at it in a more general sense than just the words.

If you look at everything that exists in the universe, we tend to forget how ‘complex’ things really are, both the living and the non living. To arrive at a thing such as an apple pie, you need to go through all that ‘complexity’: from creating the universe, to the laws of nature: the physics and the chemistry that actually structure the apple pie.
Agenda

● Hello World in C
● Hello World in C++
Hello World in C
```c
#include <stdio.h>

int main() {
    puts("Hello World!");
}
```
Magic
Translation
Source File → Preprocessor → Preprocessed source → Compiler → Assembly → Assembler → Object File → Linker → Executable
#define X Y
#define X(A) Y
#define X(A) Y
X(array<int, 4>)
#define X(A) Y
X(array<int, 4>){}
#ifdef ACCU
puts("Hello ACCU!");
#else
puts("Hi person at home!");
#endif
puts("Hello ACCU!");
#include <file>
#include “file”
Preprocessed source

Preprocessed source

Frontend

IR*

Middle-end

Optimized IR

Backend

Assembly

* IR = Intermediate Representation
Instruction Encoding (MIPS)

```
add $s1, $s2, $s3
```
Instruction Encoding (MIPS)

add $s1, $s2, $s3

<table>
<thead>
<tr>
<th>opcode</th>
<th>6 bits</th>
<th>rs</th>
<th>5 bits</th>
<th>rt</th>
<th>5 bits</th>
<th>rd</th>
<th>5 bits</th>
<th>shift</th>
<th>5 bits</th>
<th>funct</th>
<th>6 bits</th>
</tr>
</thead>
</table>

### Instruction Encoding (MIPS)

```
add $s1, $s2, $s3
```

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>18</td>
<td>19</td>
<td>17</td>
<td>0</td>
<td>0x20</td>
</tr>
</tbody>
</table>
Instruction Encoding (MIPS)

\[
\text{add } \$s1, \$s2, \$s3
\]

| 000000 | 10010 | 10011 | 10001 | 00000 | 100000 |
Instruction Encoding (MIPS)

\texttt{add \$s1, \$s2, \$s3}

\begin{verbatim}
00000010 01010011 10001000 00100000
\end{verbatim}
Assembler Directives

.data
variable_name:
.space 4
.align 2
.text
EXECUTABLE AND LINKABLE FORMAT

```
me@nux:~$ ./mini
me@nux:~$ echo $?
42
```

```
0123456789ABCDEF
00: 7F E L F 01 01 01
10: 02 00 03 00 01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
20: 34 00 20 00 01 00
40: 01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
50: 70 00 00 00 70 00 00 00 05 00 00 00
60: BB 2A 00 00 00 BB 01 00 00 00 CD 80
```

**ELF HEADER**
- **Identify as an ELF Type**
- **Specify the Architecture**

**PROGRAM HEADER TABLE**
- **Execution Information**

**MINI**

```
X86 ASSEMBLY
mov ebx, 42
mov eax, SC_EXIT
int 80h

EQUIVALENT C CODE
{return 42;}
```
ELF file types

Object File

Executable

Shared library

Core dump
ELF file types

Object File - A part of your program in bits (sections)

Executable - Your whole program as a “whole”

Shared library - Shared bits between programs

Core dump - Your whole program as a crash dump
Linker

- Take all passed-in object files
- Create lookup table of symbols referenced
Linker

- Take all passed-in object files
- Create lookup table of symbols referenced
- For each symbol not found:
  - Look through libraries to find the object file containing it
  - Load just that object file
Linker

- Take all passed-in object files
- Create lookup table of symbols referenced
- For each symbol not found:
  - Look through libraries to find the object file containing it
  - Load just that object file
- Rewrite all references in the byte code to point to actual symbols
Linker

- Take all passed-in object files
- Create lookup table of symbols referenced
- For each symbol not found:
  - Look through libraries to find the object file containing it
  - Load just that object file
- Rewrite all references in the byte code to point to actual symbols
- Output all loaded symbols and their data to an executable
Preprocessed source

Compiler

Assembly
Preprocessed source

Frontend

IR*

Middle-end

Optimized IR

Backend

Assembly

* IR = Intermediate Representation
Preprocessed source

Frontend

IR*

Middle-end

Optimized IR

Backend

Assembly

* IR = Intermediate Representation
Preprocessed source

Frontend

IR*

* IR = Intermediate Representation
Preprocessed source

Lexer

Tokens

Parser

AST*

Semantic Analyser

Annotated AST

Code Generation

IR

*AST = Abstract Syntax Tree
Preprocessed source

Lexer

Tokens

Parser

AST*

Semantic Analyser

Annotated AST

Code Generation

IR

*AST = Abstract Syntax Tree
Tokens

```c
int main() {
    puts("Hello world!");
}
```
int main() {
    puts("Hello world!");
}

• Have to deal with:
  ○ Whitespace
  ○ Identifiers
  ○ Strings
  ○ Punctuation
  ○ Multi-char operators
main() {
    puts("Hello world!");
}
int main() {
    puts("Hello world!" folk);
}

Tokens

ID(int)
ID(main)
int main() {
    puts("Hello world!");
}

Tokens
ID(int)
ID(main)
LPAREN
int main() {
    puts("Hello world!");
}

ID(int)
ID(main)
LPAREN
RPAREN
LBRACE
ID(puts)
LPAREN
STRING(Hello world!)
RPAREN
SEMI
RBRACE
Lexer Implementation

Example: recognize the tokens AB and AC
Lexer Implementation

Example: recognize the tokens AB and AC
Lexer Implementation - Switch

```cpp
while (keep_going) {
    switch(get_char()) {
        case 'A': {
            switch(get_char()) {
                case 'B': tokens.push_back(token::ab); break;
                case 'C': tokens.push_back(token::ac); break;
            }
        }
    }
}
```
Lexer Implementation - Flex

%%

AB { return token::ab; }
AC { return token::ac; }

%%
Lexer Implementation - Flex

```
%%

{letter}({letter}|{digit})*
{  yylval.id = strdup(yytext); return IDENT; }

{digit}+
{  yylval.num = atoi(yytext); return NUMBER; }

[ \t\n\r] /* skip whitespace */

. { printf("Unknown char\n"); return UNKNOWN; }

%%
```

https://en.wikipedia.org/wiki/Flex_(lexical_analyser_generator)
Preprocessed source

Lexer

Tokens

Parser

AST* (Abstract Syntax Tree)

Semantic Analyser

Annotated AST

Code Generation

IR
ID(int)
ID(main)
LPAREN
RPAREN
LBRACE
ID(puts)
LPAREN
STRING(Hello world!)
RPAREN
SEMI
RBRACE
(Extended) Bachus-Naur Form

Definition ::= Name ‘::=’ Body

Something ::= Parts That Make It ‘.’

Repetition ::= { Something }

Optional ::= [ Something ]

Result ::= ‘a’ [ { Very } Flexible ] Language [ ‘for’ Grammars ]
Type ::= ID(int) | ...

Function ::= Type Name LPAREN [ ArgDecl { ‘,’ ArgDecl } ] RPAREN LBRACE { Statement } RBRACE

Statement ::= Expression SEMI | ...

Expression ::= Name LPAREN [ Expression { ‘,’ Expression } ] RPAREN | String | ...
ID(int)
ID(main)
LPAREN
RPAREN
LBRACE
ID(puts)
LPAREN
STRING(Hello world!)
RPAREN
SEMI
RBRACE
Type ::= ID(int) | ...

Name ::= ID(...)
Type → ID(int)
Name → ID(main)
LPAREN
RPAREN
LBRACE
Name → ID(puts)
LPAREN
STRING(Hello world!)
RPAREN
SEMI
RBRACE
Function ::= Type Name LPAREN [ ArgDecl { ‘,’ ArgDecl } ] RPAREN LBRACE { Statement } RBRACE
Function ➔ Type ➔ ID(int)
  Name ➔ ID(main)
LPAREN
  <empty>
RPAREN
LBRACE
Name ➔ ID(puts)
LPAREN
STRING(Hello world!)
RPAREN
SEMI
RBRACE
Statement ::= Expression SEMI | ...
Function → Type → ID(int)
   Name → ID(main)
   LPAREN
   <empty>
   RPAREN
   LBRACE
   Statement → Name → ID(puts)
   LPAREN
   STRING(Hello world!)
   RPAREN
   SEMI
   RBRACE
Expression ::= Name LPAREN [ Arg { ‘,’ Arg } ] RPAREN | String | ...
Function → Type → ID(int)
Name → ID(main)
LPAREN
<empty>
RPAREN
LBRACE
Statement → Expression → Name → ID(puts)
LPAREN
STRING(Hello world!)
RPAREN
SEMI
RBRACE
Expression ::= Name LPAREN [ Arg { ‘,’ Arg } ] RPAREN | String | ...
Function ➔ Type ➔ ID(int)
  Name ➔ ID(main)
LPAREN
  <empty>
RPAREN
LBRACE
Statement ➔ Expression ➔ Name ➔ ID(puts)
  LPAREN
  Expression ➔ STRING(Hello world!)
  RPAREN
SEMI
RBRACE
Function

Return Type: ID(int)
Name: ID(main)
Arguments: <empty>
Statements: Call Expression
  Name: ID(puts)
  Arguments: STRING(Hello world)
Parser Implementation

selection_statement

: IF LPAREN expression RPAREN statement ELSE statement
| IF LPAREN expression RPAREN statement
| SWITCH LPAREN expression RPAREN statement
std::unique_ptr<selection_statement>
parse_selection_statement(parser_context& ctx) {
}

}
std::unique_ptr<selection_statement>
parse_selection_statement(parser_context& ctx) {
    auto type = next_token(ctx);
}
std::unique_ptr<selection_statement>
parse_selection_statement(parser_context& ctx) {
    auto type = next_token(ctx);
    if (type == token::if_) {
        
    }
}
}
std::unique_ptr<selection_statement>
parse_selection_statement(parser_context& ctx) {
    auto type = next_token(ctx);
    if (type == token::if_) {
        auto cond = parse_expression(ctx);
    }
}
}
std::unique_ptr<selection_statement>
parse_selection_statement(parser_context& ctx) {
    auto type = next_token(ctx);
    if (type == token::if_)
    {
        auto cond = parse_expression(ctx);
        auto if_stmt = parse_statement(ctx);
    }
}
}
std::unique_ptr<selection_statement>
parse_selection_statement(parser_context& ctx) {
    auto type = next_token(ctx);
    if (type == token::if_) {
        auto cond = parse_expression(ctx);
        auto if_stmt = parse_statement(ctx);
        return std::make_unique<selection_statement>(cond, if_stmt);
    }
}
std::unique_ptr<selection_statement>
parse_selection_statement(parser_context& ctx) {
  auto type = next_token(ctx);
  if (type == token::if_) {
    auto cond = parse_expression(ctx);
    auto if_stmt = parse_statement(ctx);
    return std::make_unique<selection_statement>(cond, if_stmt);
  }
  //...
}
Generator vs. Hand-Written

- **Generator**
  - Fast to get started
  - Can generate efficient parsers w/o much code
  - Grammar checker

- **Hand-Written**
  - Easier to handle and report errors
  - Easier to debug
  - Can write a faster, friendlier parser with enough work
Parser Implementation

```
selection_statement
    : IF LPAREN expression RPAREN statement ELSE statement
 | IF LPAREN expression RPAREN statement
 | SWITCH LPAREN expression RPAREN statement

if (a) if (b) puts(‘x’); else puts(‘y’);
```
Preprocessed source

Lexer

Tokens

Parser

AST*

Semantic Analyser

Annotated AST

Code Generation

IR

*AST = Abstract Syntax Tree
Preprocessed source

Lexer

Tokens

Parser

AST*

Semantic Analyser

Annotated AST

Code Generation

IR

*AST = Abstract Syntax Tree
int main() {
    auto s = "wat";
}
int main() {
    int i = "wat";
}

Preprocessed source

Lexer
Tokens
Parser
AST*
Semantic Analyser
Annotated AST
Code Generation
IR

*AST = Abstract Syntax Tree
Preprocessed source

Lexer

Tokens

Parser

AST*

Semantic Analyser

Annotated AST

Code Generation

IR

*AST = Abstract Syntax Tree
Intermediate Representation

- C++
- Rust
- Swift
- x86
- ARM
- MIPS
Intermediate Representation

- C++
- Rust
- Swift
- x86
- ARM
- MIPS
Intermediate Representation

- C++
- Rust
- Swift

- IR
- x86
- ARM
- MIPS
Intermediate Representation

- C++
- Rust
- Swift
- x86
- ARM
- MIPS
Intermediate Representation

Clang

C++
Rust
Swift

LLVM

x86
ARM
MIPS

IR
Function → Type → ID(int)
   Name → ID(main)
LPAREN
   <empty>
RPAREN
LBRACE
Statement → Expression → Name → ID(puts)
   LPAREN
   Expression →
   STRING(Hello world!)
   RPAREN
   SEMI
RBRACE
Function ➔ Type ➔ ID(int)
Name ➔ ID(main)
LBRACE
Statement ➔ Expression ➔ Name ➔ ID(puts)
   LPAREN
   Expression ➔
   STRING(Hello world!)
   RPAREN
   SEMI
RBRACE
main:
LBRACE

Statement ➔ Expression ➔ Name ➔ ID(puts)
   LPAREN
   Expression ➔ STRING(Hello
   world!)
   RPAREN

SEMI

RBRACE
main:
LBRACE

r0 = Expression → STRING(Hello world!)

Statement → Expression → Name → ID(puts)
    LPAREN
    r0
    RPAREN
    SEMI

RBRACE
main:
LBRACE

r0 = Expression ➔ STRING(Hello world!)

r1 = Expression ➔ Name ➔ ID(puts)
    LPAREN
    r0
    RPAREN

Statement ➔ r1 SEMI

RBRACE
main:
LBRACE

    r0 = STRING(Hello world!)

    r1 = Expression ➔ Name ➔ ID(puts)
       LPAREN
          r0
       RPAREN

Statement ➔ r1 SEMI

RBRACE
main:
LBRACE

   r0 = STRING(Hello world!)

   r1 = call puts(r0)

Statement ➔ r1 SEMI

RBRACE
main:
LBRACE

    r0 = STRING(Hello world!)

    r1 = call puts(r0)

    (discard r1)

RBRACE
main:
LBRACE

    r0 = STRING(Hello world!)

    r1 = call puts(r0)

RBRACE
main:
   set up stack frame
   r0 = STRING(Hello world!)
   r1 = call puts(r0)
RBRACE
main:
  set up stack frame

  r0 = STRING(Hello world!)

  r1 = call puts(r0)

  remove stack frame

  return 0
Preprocessed source

Frontend

IR*

Middle-end

Optimized IR

Backend

Assembly

* IR = Intermediate Representation
Preprocessed source

Frontend

IR* → Middle-end

Optimized IR → Backend

Assembly

* IR = Intermediate Representation
When are variables “live?”
Liveness Analysis

Live out

Live in
Liveness Analysis

Live out for I: live ins of all successors

Live in
Liveness Analysis

Live out for I: live ins of all successors

Live in for I: live outs, minus assigned variables, plus used variables
Liveness Analysis

\[
\text{Live}_{out}(I) = \bigcup_{S \in \text{succ}(I)} \text{Live}_{in}(S)
\]

\[
\text{Live}_{in}(I) = (\text{Live}_{out}(I) - \text{DEF}(I)) \cup \text{USE}(I)
\]
Liveness Analysis

\[ E = 42; \]
\[ A = 12; \]
\[ B = 32; \]
\[ C = f(B); \]
\[ D = f(A); \]
\[ f(D); \]
\[ f(C); \]
Liveness Analysis

E = ...
A = ...
B = ...

... = f(B)

C = ...

... = f(A)
D = ...

... = f(D)

... = f(C)
Liveness Analysis

def(E)
def(A)
def(B)
use(B)
def(C)
use(A)
def(D)
use(D)
use(C)
Liveness Analysis

def(E)
def(A) Live out for I: live ins of all successors
def(B)
use(B) Live in for I: live outs, minus assigned
def(C) variables, plus used variables
use(A)
def(D)
use(D)
use(C)
Live out for $I$: live ins of all successors

Live in for $I$: live outs, minus assigned variables, plus used variables

$\text{out} = \{}$
Liveness Analysis

def(E)
def(A)
def(B)
use(B)
def(C)
use(A)
def(D)
use(D)
use(C) - in = {C}, out = {}
Liveness Analysis

def(E)
def(A)
def(B)
use(B)
def(C)
use(A)
def(D)
use(D) - out = {C}
use(C) - in = {C}, out = {C}
Liveness Analysis

def(E)
def(A)
def(B)
use(B)
def(C)
use(A)
def(D)
use(D) - in = {C,D}, out = {C}
use(C) - in = {C}, out = {}
Liveness Analysis

def(E)
def(A)
def(B)
use(B)
def(C)
use(A)
def(D) - out = \{C, D\}
use(D) - in = \{C, D\}, out = \{C\}
use(C) - in = \{C\}, out = \{\}
Liveness Analysis

def(E)
def(A)  Live out for I: live ins of all successors
def(B)
use(B)  Live in for I: live outs, minus assigned variables, plus used variables
def(C)
use(A)
def(D) - in = \{C\}, out = \{C, D\}
use(D) - in = \{C, D\}, out = \{C\}
use(C) - in = \{C\}, out = \{\}
Liveness Analysis

def(E)
def(A)
def(B)
use(B)
def(C)
use(A) - in = \{A,C\}, out = \{C\}
def(D) - in = \{C\}, out = \{C,D\}
use(D) - in = \{C,D\}, out = \{C\}
use(C) - in = \{C\}, out = \{\}
Liveness Analysis

def(E)
def(A)
def(B)
use(B)
def(C) - in = \{A\}, out = \{A,C\}
use(A) - in = \{A,C\}, out = \{C\}
def(D) - in = \{C\}, out = \{C,D\}
use(D) - in = \{C,D\}, out = \{C\}
use(C) - in = \{C\}, out = {}
Liveness Analysis

def(E)
def(A)
def(B)
use(B) - in = \{A,B\}, out = \{A\}
def(C) - in = \{A\}, out = \{A,C\}
use(A) - in = \{A,C\}, out = \{C\}
def(D) - in = \{C\}, out = \{C,D\}
use(D) - in = \{C,D\}, out = \{C\}
use(C) - in = \{C\}, out = \{\}
Liveness Analysis

def(E)
def(A)
def(B) - in = \{A\}, out = \{A,B\}
use(B) - in = \{A,B\}, out = \{A\}
def(C) - in = \{A\}, out = \{A,C\}
use(A) - in = \{A,C\}, out = \{C\}
def(D) - in = \{C\}, out = \{C,D\}
use(D) - in = \{C,D\}, out = \{C\}
use(C) - in = \{C\}, out = \{\}
Liveness Analysis

def(E)
def(A) - in = {}, out = {A}
def(B) - in = {A}, out = {A,B}
use(B) - in = {A,B}, out = {A}
def(C) - in = {A}, out = {A,C}
use(A) - in = {A,C}, out = {C}
def(D) - in = {C}, out = {C,D}
use(D) - in = {C,D}, out = {C}
use(C) - in = {C}, out = {}}
Liveness Analysis

def(E) - in = {}, out = {}
def(A) - in = {}, out = {A}
def(B) - in = {A}, out = {A,B}
use(B) - in = {A,B}, out = {A}
def(C) - in = {A}, out = {A,C}
use(A) - in = {A,C}, out = {C}
def(D) - in = {C}, out = {C,D}
use(D) - in = {C,D}, out = {C}
use(C) - in = {C}, out = {}
Dead Store Elimination

def(E) - in = {}, out = {}
def(A) - in = {}, out = {A}
def(B) - in = {A}, out = {A,B}
use(B) - in = {A,B}, out = {A}
def(C) - in = {A}, out = {A,C}
use(A) - in = {A,C}, out = {C}
def(D) - in = {C}, out = {C,D}
use(D) - in = {C,D}, out = {C}
use(C) - in = {C}, out = {}}
Dead Store Elimination

def(A) - in = {}, out = {A}
def(B) - in = {A}, out = {A,B}
use(B) - in = {A,B}, out = {A}
def(C) - in = {A}, out = {A,C}
use(A) - in = {A,C}, out = {C}
def(D) - in = {C}, out = {C,D}
use(D) - in = {C,D}, out = {C}
use(C) - in = {C}, out = {}}
Preprocessed source

- Frontend
  - IR*
  - Middle-end
    - Optimized IR
    - Backend
      - Assembly

* IR = Intermediate Representation
Optimized IR

Instruction Selection

Instruction Scheduling

Register Allocation

Target-Specific Optimization

Assembly Output

Assembly
Optimized IR

- Instruction Selection
- Instruction Scheduling
- Register Allocation
- Target-Specific Optimization
- Assembly Output

Assembly
Instruction Selection

```c
int* v0, v1;

*v0 = *v0 + *v1;
```
Instruction Selection

\[ r0 = \text{load} \ v0 \]
\[ r1 = \text{load} \ v1 \]
\[ r2 = \text{add} \ r0 \ r1 \]
\[ \text{store} \ v0 \ r2 \]
Macro Expansion

\[ r_0 = \text{load} \ v_0 \quad \text{mov} \ eax, [rsi] \]

\[ r_1 = \text{load} \ v_1 \quad \text{mov} \ ebx, [rdi] \]

\[ r_2 = \text{add} \ r_0 \ r_1 \quad \text{add} \ eax, ebx \]

\[ \text{store} \ v_0 \ r_2 \quad \text{mov} \ [rsi], eax \]
Macro Expansion

\[
\begin{align*}
  r0 &= \text{load } v0 \\
  r1 &= \text{load } v1 \\
  r2 &= \text{add } r0 \, r1 \\
  \text{store } v0 \, r2
\end{align*}
\]
Selection DAG

\[ r_0 = \text{load} \ v_0 \]
\[ r_1 = \text{load} \ v_1 \]
\[ r_2 = \text{add} \ r_0 \ r_1 \]
\[ \text{store} \ v_0 \ r_2 \]
Selection DAG

```
store
 / \    / \  
v0   add  load
   /    /    /
  v0  v0   v1
```
Selection DAG

store

v0  add

load  load

v0  v1
Selection DAG

add [rdi], eax
Selection DAG

store

\( \text{v0} \)

add

load

\( \text{v0} \)

load

\( \text{v1} \)

add \([\text{rdi}], \text{eax}\)
mov eax, [rsi]
add [rdi], eax
Optimized IR

Instruction Selection

Instruction Scheduling

Register Allocation

Target-Specific Optimization

Assembly Output

Assembly
Instruction Scheduling

add r8, r9
add r8, r10
add r11, r12
add r11, r13
Instruction Scheduling

add r8, r9
add r11, r12
add r8, r10
add r11, r13
Graph Colouring
Graph Colouring
Interference Graph

def(E) - in = {}, out = {}
def(A) - in = {}, out = {A}
def(B) - in = {A}, out = {A,B}
use(B) - in = {A,B}, out = {A}
def(C) - in = {A}, out = {A,C}
use(A) - in = {A,C}, out = {C}
def(D) - in = {C}, out = {C,D}
use(D) - in = {C,D}, out = {C}
use(C) - in = {C}, out = {}}
Interference Graph

def(E) - in = {}, out = {}
def(A) - in = {}, out = {A}
def(B) - in = {A}, out = {A,B}
use(B) - in = {A,B}, out = {A}
def(C) - in = {A}, out = {A,C}
use(A) - in = {A,C}, out = {C}
def(D) - in = {C}, out = {C,D}
use(D) - in = {C,D}, out = {C}
use(C) - in = {C}, out = {}
Interference Graph

def(E) - in = {}, out = {}
def(A) - in = {}, out = {A}
def(B) - in = {A}, out = {A,B}
use(B) - in = {A,B}, out = {A}
def(C) - in = {A}, out = {A,C}
use(A) - in = {A,C}, out = {C}
def(D) - in = {C}, out = {C,D}
use(D) - in = {C,D}, out = {C}
use(C) - in = {C}, out = {C}
Interference Graph

def(E) - in = {}, out = {}
def(A) - in = {}, out = {A}
def(B) - in = {A}, out = {A,B}
use(B) - in = {A,B}, out = {A}
def(C) - in = {A}, out = {A,C}
use(A) - in = {A,C}, out = {C}
def(D) - in = {C}, out = {C,D}
use(D) - in = {C,D}, out = {C}
use(C) - in = {C}, out = {}}
E = ...
A = ...
B = ...
... = f(B)
C = ...
... = f(A)
D = ...
... = f(D)
... = f(C)
Interference Graph

E = ...
A = ...
B = ...
... = f(B)
C = ...
... = f(A)
D = ...
... = f(D)
... = f(C)
Interference Graph

E = ...
A = ...
B = ...
... = f(B)
C = ...
... = f(A)
D = ...
... = f(D)
... = f(C)

R₀ = ...
R₁ = ...
... = f(R₁)
... = f(R₀)
R₀ = ...
... = f(R₀)
... = f(R₁)
Interference Graph

E = ...
A = ...
B = ...
... = f(B)
C = ...
... = f(A)
D = ...
... = f(D)
... = f(C)

R₀ = ...
R₁ = ...
... = f(R₁)
R₁ = ...
... = f(R₀)
R₀ = ...
... = f(R₀)
... = f(R₁)
Interference Graph

\[ E = \ldots \]
\[ A = \ldots \]
\[ B = \ldots \]
\[ \ldots = f(B) \]
\[ C = \ldots \]
\[ D = \ldots \]
\[ \ldots = f(A) \]
\[ \ldots = f(D) \]
\[ \ldots = f(C) \]
Interference Graph

E = ...
A = ...
B = ...
... = f(B)
C = ...
D = ...
... = f(A)
... = f(D)
... = f(C)
mov eax, 0
mov eax, 0
b8000000000000
xor rax, rax
4831c0
xor eax, eax
31 c0
main:
set up stack frame

r0 = STRING(Hello world!)

r1 = call puts(r0)

remove stack frame
return 0
hello:
    .string “Hello World!”
main:
    set up stack frame
    r0 = OFFSET FLAT :hello
    r1 = call puts(r0)
    remove stack frame
    return 0
hello:
    .string "Hello World!"
main:
    set up stack frame

    mov rdi, OFFSET FLAT :hello

    r1 = call puts (implicit: rdi)

    remove stack frame
    return 0
hello:
    .string “Hello World!”
main:
    set up stack frame

mov rdi, OFFSET FLAT :hello

call puts

remove stack frame
return 0
hello:
    .string "Hello World!"
main:
    sub rsp, 8

    mov rdi, OFFSET FLAT :hello

    call puts

    add rsp, 8
    return 0
hello:
  .string “Hello World!”
main:
  sub rsp, 8
  mov rdi, OFFSET FLAT :hello
  call puts
  add rsp, 8
  xor eax, eax
  ret
hello:
    .string "Hello World!"
main:
    sub rsp, 8
    mov rdi, OFFSET FLAT :hello
    call puts
    add rsp, 8
    xor eax, eax
    ret

https://godbolt.org/z/AejcvA
hello:
  .string “Hello World!”
main:
  sub rsp, 8
  mov rdi, OFFSET FLAT :hello
  call puts
  add rsp, 8
  xor eax, eax
  ret

https://godbolt.org/z/AejcvA
hello:
    .string “Hello World!”
main:
    sub rsp, 8
    mov rdi, OFFSET FLAT :hello
    call puts
    add rsp, 8
    xor eax, eax
    ret

https://godbolt.org/z/AejcvA
hello:
  .string “Hello World!”
main:
  sub rdi, 8
  mov rdi, OFFSET FLAT :hello
  call puts
  add rdi, 8
  xor eax, eax
  ret

https://godbolt.org/z/AejcvA
48 65 6c 6c 6f 20 57 6f 72 6c 64 21 00

48 83 ec 08
bf 00 00 00 00
e8 00 00 00 00
e8 83 c4 08
31 c0
cb
48 65 6c 6c 6f 20 57 6f 72 6c 64 21 00

48 83 ec 08
bf 00 00 00 00 ← Relocation
e8 00 00 00 00
48 83 c4 08
31 c0
cb
48 65 6c 6c 6f 20 57 6f 72 6c 64 21 00

48 83 ec 08 bf 00 00 00 00 e8 00 00 00 00 00 48 83
c4 08 31 c0 cb
.rodata: 48 65 6c 6c 6f 20 57 6f 72 6c 64 21 00

.text: 48 83 ec 08 bf 00 00 00 00 e8 00 00 00 00 00 48 83 c4 08 31 c0 cb
.rodata: 48 65 6c 6c 6f 20 57 6f 72 6c 64 21 00

.text: 48 83 ec 08 bf 00 00 00 00 e8 00 00 00 00
00 48 83 c4 08 31 c0 cb

symtab:
  hello: .rodata + 0
  main: .text + 0
  puts: ?
.rodata: 48 65 6c 6c 6f 20 57 6f 72 6c 64 21 00

.text: 48 83 ec 08 bf 00 00 00 00 e8 00 00 00 00 48 83 c4 08 31 c0 cb

symtab:
  hello: .rodata + 0
  main: .text + 0
  puts: ?

.rel.text:
  +5: 4 bytes signed offset, point to hello
  +10: 4 bytes signed offset, point to puts
.rodata: 48 65 6c 6c 6f 20 57 6f 72 6c 64 21 00

.text: 48 83 ec 08 bf 00 00 00 00 e8 00 00 00 00 48 83 c4 08 31 c0 cb

symtab:
  hello: .rodata + 0
  main: .text + 0
  puts: ?

.rel.text:
  +5: R_X86_64_PC32, point to hello
  +10: R_X86_64_PC32, point to puts
.rodata: 48 65 6c 6c 6f 20 57 6f 72 6c 64 21 00

.text: 48 83 ec 08 bf 00 00 00 00 e8 00 00 00 00 48 83 c4 08 31 c0 cb

symtab:
    hello: .rodata + 0
    main: .text + 0
    puts: ?

.rel.text:
    +5: R_X86_64_PC32, point to hello
    +10: R_X86_64_PC32, point to puts
.rodata: 48 65 6c 6c 6f 20 57 6f 72 6c 64 21 00

.text: 48 83 ec 08 bf 00 00 00 00 e8 00 00 00 00 48 83 c4 08 31 c0 cb

.puts.o’s) .text: 41 55 41 54 49 89 fc 55 53 48 83 ec 08 ...

.puts.o) symtab: puts: .text + 0x0
.rodata: 48 65 6c 6c 6f 20 57 6f 72 6c 64 21 00

.text: 48 83 ec 08 bf 00 00 00 00 e8 00 00 00 00 48 83 c4 08 31 c0 cb

(puts.o’s) .text: 41 55 41 54 49 89 fc 55 53 48 83 ec 08 ...

(puts.o) symtab: puts: .text + 0x0
.rodata: 48 65 6c 6c 6f 20 57 6f 72 6c 64 21 00

.text: 48 83 ec 08 bf 00 00 00 00 e8 00 00 00 00
00 48 83 c4 08 31 c0 cb 41 55 41 54 49 89 fc 55
53 48 83 ec 08 ...

symtab:
    hello: .rodata + 0
    main: .text + 0
    puts: .text + 0x15

.rel.text:
    +5: R_X86_64_PC32, point to hello
    +10: R_X86_64_PC32, point to puts
```assembly
48 65 6c 6c 6f 20 57 6f 72 6c 64 21 00 48 83 ec 08 bf 00 00 00 00 e8 00 00 00 00 48 83 c4 08 31 c0 cb 41 55 41 54 49 89 fc 55 53 48 83 ec 08 ... 

symtab:
  hello: 8048000 + 0x0
  main: 804800d + 0
  puts: 804800d + 0x15

.rel.text:
  +5: R_X86_64_PC32, point to hello
  +10: R_X86_64_PC32, point to puts
```
48 65 6c 6c 6f 20 57 6f 72 6c 64 21 00 48 83 ec 08 bf 00 00 00 00 e8 00 00 00 00 48 83 c4 08 31 c0 cb 41 55 41 54 49 89 fc 55 53 48 83 ec 08 ...

symtab:
    hello: 8048000
    main: 804800d
    puts: 8048022
.rel.text:
     8048012: R_X86_64_PC32, point to hello
     8048017: R_X86_64_PC32, point to puts
48 65 6c 6c 6f 20 57 6f 72 6c 64 21 00 48 83 ec 08 bf 00 00 00 00 e8 00 00 00 00 00 48 83 c4 08 31 c0 cb 41 55 41 54 49 89 fc 55 53 48 83 ec 08 ...

symtab:
    hello: 8048000
    main: 804800d
    puts: 8048022
.rel.text:
    8048012: R_X86_64_PC32, point to hello
    8048017: R_X86_64_PC32, point to puts
symtab:
  hello: 8048000
  main: 804800d
  puts: 8048022
.rel.text:
  8048012: R_X86_64_PC32, 0xffffffffffef
  8048017: R_X86_64_PC32, 0x00000000c
symtab:
  hello: 8048000
  main: 804800d
  puts: 8048022
48 65 6c 6c 6f 20 57 6f 72 6c 64 21 00 48 83 ec 08 bf ef ff ff ff e8 0c 00 00 00 48 83 c4 08 31 c0 cb 41 55 41 54 49 89 fc 55 53 48 83 ec 08 ...

symtab:
  main: 804800d

Program table:
  Load these bytes at 0x8048000

Elf header:
  Entry point is 0x804800d
Program table:
Load these bytes at 0x8048000

Elf header:
Entry point is 0x804800d
Hello world!
Hello World in C++
#include <iostream>

int main() {
    std::cout << "Hello World!\n";
}
a<b<x>>y;
a<b < x>>y;
a < b < x > y ;
a < b << x >> > y ;
std::cout
class std::basic_ostream<char>
(class std::basic_ios<char> (virtual base)
    class std::ios_base (primary base)
    (ios_base vtable pointer)
    std::streamsize _M_precision
    std::streamsize _M_width
    std::ios_base::fmtflags _M_flags
    std::ios_base::iostate _M_exception
    std::ios_base::iostate _M_streambuf_state
    struct std::ios_base::Callback_list * _M_callbacks
    struct std::ios_base::Words _M_word_zero
    void * _M_pword
    long _M_iword
    struct std::ios_base::Words [8] _M_local_word
    int _M_word_size
    struct std::ios_base::Words * _M_word
    class std::locale _M_ios_locale
    class std::locale::_Impl * _M_impl
    basic_ostream<char, struct std::char_traits<char>> * _M_tie
    std::basic_ios<char, struct std::char_traits<char>>::char_type _M_fill
    _Bool _M_fill_init
    basic_streambuf<char, struct std::char_traits<char>> * _M_streambuf
    const std::basic_ios<char, struct std::char_traits<char>>::_ctype_type * _M CType
    const std::basic_ios<char, struct std::char_traits<char>>::_num_put_type * _M_num_put
    const std::basic_ios<char, struct std::char_traits<char>>::_num_get_type * _M_num_get
Use fmtlib
Use fmtlib
(P0645, C++20)
struct weird_int {
    int i;
    virtual void get_value(){}
};
struct weird_int {
    int i;
    virtual void get_value() {}
};

0 | struct weird_int
0 | (weird_int vtable pointer)
8 | int i
vtable

vtable ptr

int i
vtable

- vtable ptr
- int i
- offset-to-top
- RTTI ptr
- get_value ptr
vtable

int i

vtable ptr

weird_int

offset-to-top

RTTI ptr

get_value ptr

weird_int::get_value

RTTI
vtable

郜

int i

vtable ptr

_offset-to-top

RTTI ptr

get_value ptr

_ZTS9weird_int

weird_int

_ZTI9weird_int

_ZTV9weird_int

weird_int RTTI

weird_int::

get_value

_ZN9weird_int9get_valueEv
_ZTS9weird_int
  .string "weird_int"

_ZN9weird_int9get_valueEv:
  mov eax, DWORD PTR [edi+8]
  ret
Object Lifetimes

- **Static**
  - Global scope
  - Function scope
- **Thread local**
- **Dynamic (heap)**
- **Automatic (stack)**
ELF file types

Object File - A part of your program in bits (sections)

Executable - Your whole program as a “whole”

Shared library - Shared bits between programs

Core dump - Your whole program as a crash dump
Shared library

- Contains a ton of functions you can import
- Read-only and code are shared between all processes
- Allows security patches without full recompile
- Origin of DLL Hell
Shared library

- What if symbol names collide?
  - Use from first loaded executable/library
Shared library

- What if symbol names collide?
  - Use from first loaded executable/library

puts

puts@plt (points to puts)

main (points to puts@plt)
Shared library

- What if symbol names collide?
  - Use from first loaded executable/library

```
puts

puts@plt (points to puts)

main (points to puts@plt)
```

```
puts

puts@plt

fputs (points to puts@plt)
```
Shared library

- What if symbol names collide?
  - Use from first loaded executable/library

```
puts

puts@plt (points to puts)

main (points to puts@plt)
```

```
puts

puts@plt (points to puts)

fputs (points to puts@plt)
```
Shared library

- What if symbol names collide?
  - Use from first loaded executable/library

`puts@plt` (points to `puts`)  
`puts@plt` (points to `puts`)  
`main` (points to `puts@plt`)  
`fputs` (points to `puts@plt`)
Shared library

- What if symbol names collide?
  - Use from first loaded executable/library
- PLT (or equivalent) for functions
  - Procedure Linkage Table
- GOT (or equivalent) for objects
  - Global Offset Table
- Often, the PLT uses the GOT.
Global initializer(s)

```c
struct T {
    T() {}
    ~T() {}
};
T t;
```
Global initializer(s)

_GLOBAL__sub_I_t:
  mov edi,0x601044 (address of t)
call 400626 "T::T()" (construct T)

  mov edx,0x601038 (address of __dso_handle)
  mov esi,0x601044 (address of t)
  mov edi,0x4005f2 (address of T::~T)
  jmp 4004e0 "__cxa_atexit@plt"
symtab:

main: 804800d

Program table:

Load these bytes at 0x8048000

Elf header:

Entry point is 0x804800d
48 65 6c 6c 6f 20 57 6f 72 6c 64 21 00 48 83 ec 08 bf ef ff ff ff e8 0c 00 00 00 48 83 c4 08 31 c0 cb 41 55 41 54 49 89 fc 55 53 48 83 ec 08 ...

symtab:
  main: 804800d
  _start: 8048432

Program table:
  Load these bytes at 0x8048000

Elf header:
  Entry point is 0x8048432
● Crtbegin.o
  ○ __dso_handle in .data
  ○ __frame_dummy_init_array_entry in .init_array
● Crtend.o
● Linker trick
  ○ creates an array of global initializers
  ○ creates global variables pointing to the start and end
● Link in _start to call these global initializers
_start:
    xor  %ebp,%ebp
    pop  %rsi
    mov  %rsp,%r9
    and  $0xfffffffffffffff0,%rsp
    push %rax
    push %rsp
    lea  0x16a(%rip),%r8      # 11d0 <__libc_csu_fini>
    lea  0x103(%rip),%rcx     # 1170 <__libc_csu_init>
    lea  0xc3(%rip),%rdi      # 1137 <main>
    callq *0x2f66(%rip)       # 3fe0 <__libc_start_main>
    hlt
Loader
ELF Loader

- Loads your ELF executable from disk
  - Loads any required shared libraries first
  - Causes globals to be initialized for shared libraries
ELF Loader

- Loads your ELF executable from disk
  - Loads any required shared libraries first
  - Causes globals to be initialized for shared libraries
- Handles “where is our `puts` now”
- Handles `__cxa_atexit` and `__dso_handle` complexities
- PLT entries are all empty, pointing to the loader
./hello
Hello world!
Summary / conclusion
This was

Hello World from Scratch

Peter Bindels
he/him
@dascandy42
Principal Software Engineer
TomTom

Simon Brand
they/them
@tartanllama
C++ Developer Advocate
Microsoft
Dragon Book
Dragon Book
or can also come up when a student...

THAT'S ALL FOLKS.

Jessica Paquette
@barrelshifter  Follows you
https://llvm.org/docs/tutorial/
https://llvm.org/docs/WritingAnLLVMBackend.html