Good Fun: Creating a Data-Oriented Parser/AST/Visitor Generator

Dr. Robert Schadek

DConf 2024

Why

- I like to writing parser generators

Why

- I like to writing parser generators
  - I do not need them
  - I do not like to use them for something useful
- but they are really good fun

Yacc

```
expr : expr '+' expr
  {
     $$ = node( '+', $1, $3 );
  }
```

A bit of Darser history

Darser is a parser generator that

- generates a recursive decent parser
- generates AST classes for parser to use
- generates visitor to traverse, simply inherit
- is used in graphqld

Input

```
Definition:
    O: [OperationDefinition#op]
    F: [FragmentDefinition#frag]
    T: [TypeSystemDefinition#type]
```

AST 1/3

```
class Definition : Node {
@safe :

  DefinitionEnum ruleSelection;
  FragmentDefinition frag;
  TypeSystemDefinition type;
  OperationDefinition op;
```

AST 2/3

```
this(DefinitionEnum ruleSelection, OperationDefinition op) {
  this.ruleSelection = ruleSelection;
  this.op = op;
}

this(DefinitionEnum ruleSelection, FragmentDefinition frag) {
  this.ruleSelection = ruleSelection;
  this.frag = frag;
}

this(DefinitionEnum ruleSelection, TypeSystemDefinition type) {
  this.ruleSelection = ruleSelection;
  this.type = type;
}
```

AST 3/3

```
  void visit(Visitor vis) {
    vis.accept(this);
  }

  void visit(Visitor vis) const {
    vis.accept(this);
  }

  void visit(ConstVisitor vis) {
    vis.accept(this);
  }

  void visit(ConstVisitor vis) const {
    vis.accept(this);
  }
}
```

Parser Example 1 1/2

```
Definition parseDefinitionImpl() {
  if(this.firstOperationDefinition()) {
    OperationDefinition op = this.parseOperationDefinition();

    return new Definition(DefinitionEnum.O, op);
  } else if(this.firstFragmentDefinition()) {
    FragmentDefinition frag = this.parseFragmentDefinition();

    return new Definition(DefinitionEnum.F, frag);
  } else if(this.firstTypeSystemDefinition()) {
    TypeSystemDefinition type = this.parseTypeSystemDefinition();

    return new Definition(DefinitionEnum.T, type);
  }
  auto app = appender!string();
  formattedWrite(app,
    "In 'Definition' found a '%s' while looking for",
    this.lex.front
  );
  throw new ParseException(app.data,
```

Parser Example 1 2/2

```
bool firstOperationType() const {
  return this.lex.front.type == TokenType.query
     || this.lex.front.type == TokenType.mutation
     || this.lex.front.type == TokenType.subscription;
}
```

Visitor

```
class Visitor : ConstVisitor {
  void enter(Definition obj) {}
  void exit(Definition obj) {}

  void accept(Definition obj) {
    enter(obj);
    final switch(obj.ruleSelection) {
      case DefinitionEnum.O:
        obj.op.visit(this);
        break;
      case DefinitionEnum.F:
        obj.frag.visit(this);
        break;
      case DefinitionEnum.T:
        obj.type.visit(this);
        break;
    }
    exit(obj);
  }
}
```

Visitor Usage

```
class CountVisitor : ConstVisitor {
  void accept(Definition obj) {
    super.accept(obj);
    this.definitionCnt++;
  }
}
```

Input 2

```
InlineFragment:
    TDS: [on_, name#tc, Directives#dirs, SelectionSet#ss]
    TS: [on_, name#tc, SelectionSet#ss]
    DS: [Directives#dirs, SelectionSet#ss]
    S: [SelectionSet#ss]
```

Parser Example 2 1/2

```
InlineFragment parseInlineFragmentImpl() {
 if(this.lex.front.type == TokenType.on_) {
  this.lex.popFront();
  if(this.lex.front.type == TokenType.name) {
   Token tc = this.lex.front;
   this.lex.popFront();
   if(this.firstDirectives()) {
    Directives dirs = this.parseDirectives();
    if(this.firstSelectionSet()) {
     SelectionSet ss = this.parseSelectionSet();

     return new InlineFragment(InlineFragmentEnum.TDS, tc, dirs, ss);
    }
    throw new ParseException(["lcurly"]);
   } else if(this.firstSelectionSet()) {
    SelectionSet ss = this.parseSelectionSet();

    return new InlineFragment(InlineFragmentEnum.TS, tc, ss);
   }
```

Parser Example 2 2/2

```
      throw new ParseException(["at -> Directive","lcurly"]);
    }
    throw new ParseException(["name"]);
  } else if(this.firstDirectives()) {
    Directives dirs = this.parseDirectives();
    if(this.firstSelectionSet()) {
      SelectionSet ss = this.parseSelectionSet();

      return new InlineFragment(InlineFragmentEnum.DS, dirs, ss);
    }
    throw new ParseException(["lcurly"]);
  } else if(this.firstSelectionSet()) {
    SelectionSet ss = this.parseSelectionSet();
    return new InlineFragment(InlineFragmentEnum.S, ss);
  }
  throw new ParseException(["on_","at -> Directive","lcurly"]);
}
```

Data-oriented Design (DoD)

Data-oriented Design (DoD)

Putting data that is accessed together in arrays, while making sure that every bit counts!

Hardware

	time	size
INST	$\approx$ 0.25-10 cycles	128B

Hardware

	time	size
INST	≈ 0.25-10 cycles	128B
L1	3 cycles	16KB - 128 KB
L2	10 cycles	256KB - 1MB
L3	40 cycles	2MB - 32MB
RAM	100 cycles	how much money do you have

Why use Arrays

- L1 cache lines are loaded one line at a time
- chances are good that after reading one array element you read the next
- Pointers on 64bit system are wasteful
- At least on current 64bit Linux you can only address 2^{48} bit.

Why use Arrays

- L1 cache lines are loaded one line at a time
- chances are good that after reading one array element you read the next
- Pointers on 64bit system are wasteful
- At least on current 64bit Linux you can only address 2^{48} bit.

- If an `uint` index is not good enough, reconsider your decisions

What now

- Its called Abstract Syntax Tree not Abstract Syntax Array

What now

- Its called Abstract Syntax Tree not Abstract Syntax Array

- But what is an Tree with Nodes and Pointers then indices into the ultimate array that is main memory.

What now

- Its called Abstract Syntax Tree not Abstract Syntax Array
- But what is an Tree with Nodes and Pointers then indices into the ultimate array that is main memory.
- How hard can it be

AST Array

```
struct OperationDefinition {
  uint vdIdx;
  uint otIdx;
  uint dIdx;
  uint ssIdx;
  Token name;
  OperationDefinitionEnum ruleSelection;
```

Parser Array 1/3

```
struct Parser {
  Document[] documents;
  Definitions[] definitionss;
  Definition[] definitions;
  OperationDefinition[] operationDefinitions;
  SelectionSet[] selectionSets;
  OperationType[] operationTypes;
  Selections[] selectionss;
  Selection[] selections;
  FragmentSpread[] fragmentSpreads;
  InlineFragment[] inlineFragments;
  Field[] fields;
  FieldName[] fieldNames;
  Arguments[] argumentss;
  ArgumentList[] argumentLists;
  Argument[] arguments;
```

Parser Array 2/3

```
uint parseOperationDefinitionImpl() {
  string[] subRules;
  if(this.firstSelectionSet()) {
    uint ss = this.parseSelectionSet();

    this.operationDefinitions ~= OperationDefinition.ConstructSelSet(ss);
    return cast(uint)(this.operationDefinitions.length - 1);

  } else if(this.firstOperationType()) {
    uint ot = this.parseOperationType();
    if(this.lex.front.type == TokenType.name) {
      Token name = this.lex.front;
      this.lex.popFront();
      if(this.firstVariableDefinitions()) {
        uint vd = this.parseVariableDefinitions();
```

Parser Array 3/3

```
    uint vd = this.parseVariableDefinitions();
    if(this.firstDirectives()) {
      uint d = this.parseDirectives();
      if(this.firstSelectionSet()) {
        uint ss = this.parseSelectionSet();

        this.operationDefinitions ~= OperationDefinition.ConstructOT_N_VD(ot, name, vd, d, ss);
        return cast(uint)(this.operationDefinitions.length - 1);

      }
```

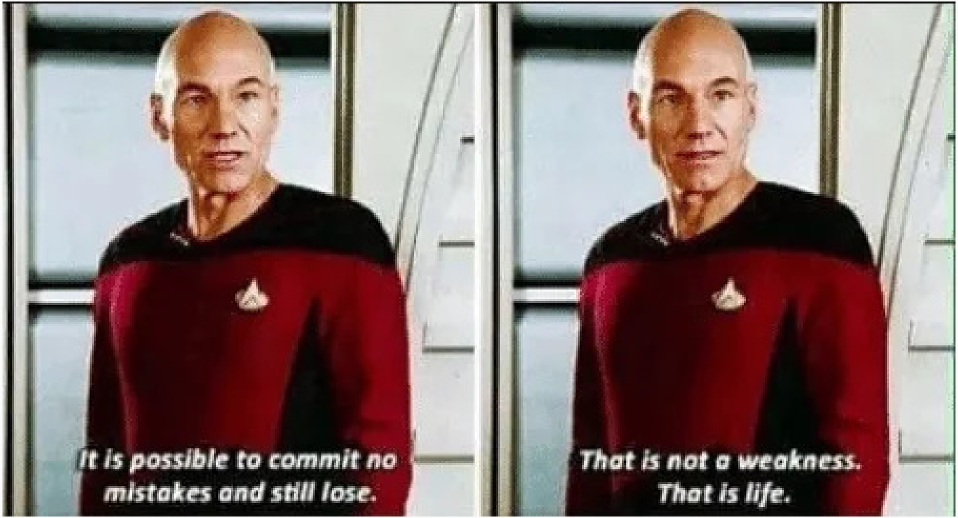
Visitor Array 1/3

```
void accept(ref OperationDefinition obj) {
  enter(obj);
  final switch(obj.ruleSelection) {
    case OperationDefinitionEnum.SelSet:
      this.accept(this.parser.selectionSets[obj.ssIdx]);
      break;
    case OperationDefinitionEnum.OT_N_VD:
      this.accept(this.parser.operationTypes[obj.otIdx]);
      obj.name.visit(this);
      this.accept(this.parser.variableDefinitionss[obj.vdIdx]);
      this.accept(this.parser.directivess[obj.dIdx]);
      this.accept(this.parser.selectionSets[obj.ssIdx]);
      break;
    case OperationDefinitionEnum.OT_N_V:
      this.accept(this.parser.operationTypes[obj.otIdx]);
      obj.name.visit(this);
```

Results

Measure	class based	struct based
Wall Clock	5.8s	6.8s
L1-dcache-loads	10_092_429_449	10_949_701_377
L1-dcache-load-misses	141_966_518	200_291_333
L1-misses-percentage	1.4%	1.8%
Maximum resident set size	278_912 KiB	192_256 KiB

Results


It is possible to commit no mistakes and still lose.
That is not a weakness. That is life.

Structured Ranting

AST Re-Structuring

```
OperationDefinition:
    SelSet: [SelectionSet#ss]
    OT_N_VD: [OperationType#ot, name#name, VariableDefinitions#vd, Directives#d, SelectionSet#ss]
    OT_N_V: [OperationType#ot, name#name, VariableDefinitions#vd, SelectionSet#ss]
    OT_N_D: [OperationType#ot, name#name, Directives#d, SelectionSet#ss]
    OT_N: [OperationType#ot, name#name, SelectionSet#ss]
    OT_VD: [OperationType#ot, VariableDefinitions#vd, Directives#d, SelectionSet#ss]
    OT_V: [OperationType#ot, VariableDefinitions#vd, SelectionSet#ss]
    OT_D: [OperationType#ot, Directives#d, SelectionSet#ss]
    OT: [OperationType#ot, SelectionSet#ss]
```

AST Re-Structuring

```
struct OperationDefinitionEnumFirst {
  OperationDefinitionEnum ruleSelection : 4;
  uint vdIdx : 28;
}

struct OperationDefinition {
  OperationDefinitionEnumFirst vdIdx;
  uint otIdx;
  uint dIdx;
  uint ssIdx;
  uint name;
}
```

Reading/Writing AST on Disk 1/2

```
void toDisk(ref File file) {
  static foreach(mem; __traits(allMembers, Parser)) {{
    alias T = typeof(__traits(getMember, Parser, mem));
    static if(isArray!(T)) {
      file.write(cast(uint)__traits(getMember, this, mem).length);
      file.rawWrite(__traits(getMember, this, mem));
    }
  }}
}
```

Reading/Writing AST on Disk 2/2

```d
void fromDisk(ref File file) {
  static foreach(mem; __traits(allMembers, Parser)) {{
    alias T = typeof(__traits(getMember, Parser, mem));
    static if(isArray!(T)) {
      ubyte[4] lenA;
      file.rawRead(lenA[]);
      uint len = *(cast(uint*)lenA.ptr);
      T[] arr = new T[len];
      file.rawRead(arr);
      __traits(getMember, this, mem) = arr;
    }
  }}
}
```

Lexer and Tokens

Lexers and Tokens are no fun ... so much manual work

TokenType

```
enum TokenType {
    undefined
  , exclamation
  , dollar
  , lparen
  ...
}

struct Token {
  string value;
  uint line;
  uint column;
  TokenType type;
}
```

TokenType

```
struct Token {
  TokenType type : 7;
  uint valueOrIndex : 25;
}

struct TokenPos {
  uint line;
  uint column;
}

struct Lexer {
  int[] ints;
  float[] floats;
  double[] doubles;
  TokenPos[16] positions;
}
```

Lexer and Tokens

```
mutation MutateCreatePerson($legalName: LegalNameIn!
    , $knownAsName: KnownAsNameIn!
    , $privateContact: PrivateContactIn!
    , $activeAfter: DateTime
    , $includedInHeadcount: Boolean!) {
  createPerson(legalName: $legalName
      , knownAsName: $knownAsName
      , privateContact: $privateContact
      , activeAfter: $activeAfter
      , includedInHeadcount: $includedInHeadcount) {
    id
  }
}
```

That graphql only contains 20 strings that need storing

String/Array Intering

```
struct SmallStringPtr {
  uint idx;
  uint length;
}
struct StringIntering {
  string str;
  SmallStringPtr[] index;
  uint[string] map;

  uint insert(string s) {
    uint* alreadyInMap = s in this.map;
    if(alreadyInMap != null) return *alreadyInMap;
    SmallStringPtr ptr;
    ptr.index = cast(uint)this.str.length;
    ptr.length = cast(uint)s.length;
    this.str ~= s;
    uint ret = cast(uint)this.index.length;
    this.index ~= ptr;
    return ret;
```

String/Array Intering

```
struct SmallStringPtr {
  uint idx;
  uint length;
}
struct StringIntering {
  string str;
  SmallStringPtr[] index;
  uint[string] map;

  uint insert(string s) {
    uint* alreadyInMap = s in this.map;
    if(alreadyInMap != null) return *alreadyInMap;
    SmallStringPtr ptr;
    ptr.index = cast(uint)this.str.length;
    ptr.length = cast(uint)s.length;
    this.str ~= s;
    uint ret = cast(uint)this.index.length;
    this.index ~= ptr;
    return ret;
```

- Easy to read and write to file
- Initial construction slow, reading, comparison really fast
- `const StringIntering` makes `const` useful

Coming to an End

Conclusion

- Measure first

- Parser/AST/Visitor generation is fun
- DoD is not new, look at any C program from 1990
- Think Database-Normalization more often
- Looking into the past for inspiration

- `https://github.com/burner/Darser`

The End