

# 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

```
1  expr : expr '+' expr
2      {
3          $$ = node( '+', $1, $3 );
4      }
```

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 graphql

```
1 Definition:
2     O: [OperationDefinition#op]
3     F: [FragmentDefinition#frag]
4     T: [TypeSystemDefinition#type]
```

```
1  class Definition : Node {  
2    @safe :  
3  
4    DefinitionEnum ruleSelection;  
5    FragmentDefinition frag;  
6    TypeSystemDefinition type;  
7    OperationDefinition op;
```

```
1  this(DefinitionEnum ruleSelection, OperationDefinition op) {
2      this.ruleSelection = ruleSelection;
3      this.op = op;
4  }
5
6  this(DefinitionEnum ruleSelection, FragmentDefinition frag) {
7      this.ruleSelection = ruleSelection;
8      this.frag = frag;
9  }
10
11 this(DefinitionEnum ruleSelection, TypeSystemDefinition type) {
12     this.ruleSelection = ruleSelection;
13     this.type = type;
14 }
```



```
1  void visit(Visitor vis) {
2      vis.accept(this);
3  }
4
5  void visit(Visitor vis) const {
6      vis.accept(this);
7  }
8
9  void visit(ConstVisitor vis) {
10     vis.accept(this);
11 }
12
13 void visit(ConstVisitor vis) const {
14     vis.accept(this);
15 }
16 }
```

## Parser Example 1 1/2

```
1  Definition parseDefinitionImpl() {
2      if(this.firstOperationDefinition()) {
3          OperationDefinition op = this.parseOperationDefinition();
4
5          return new Definition(DefinitionEnum.O, op);
6      } else if(this.firstFragmentDefinition()) {
7          FragmentDefinition frag = this.parseFragmentDefinition();
8
9          return new Definition(DefinitionEnum.F, frag);
10     } else if(this.firstTypeSystemDefinition()) {
11         TypeSystemDefinition type = this.parseTypeSystemDefinition();
12
13         return new Definition(DefinitionEnum.T, type);
14     }
15     auto app = appender!string();
16     formattedWrite(app,
17         "In 'Definition' found a '%s' while looking for",
18         this.lex.front
19     );
20     throw new ParseException(app.data,
```

## Parser Example 1 2/2

```
1  bool firstOperationType() const {  
2      return this.lex.front.type == TokenType.query  
3          || this.lex.front.type == TokenType.mutation  
4          || this.lex.front.type == TokenType.subscription;  
5  }
```

# Visitor

```
1  class Visitor : ConstVisitor {
2      void enter(Definition obj) {}
3      void exit(Definition obj) {}
4
5      void accept(Definition obj) {
6          enter(obj);
7          final switch(obj.ruleSelection) {
8              case DefinitionEnum.O:
9                  obj.op.visit(this);
10                 break;
11             case DefinitionEnum.F:
12                 obj.frag.visit(this);
13                 break;
14             case DefinitionEnum.T:
15                 obj.type.visit(this);
16                 break;
17         }
18         exit(obj);
19     }
20 }
```

# Visitor Usage

```
1  class CountVisitor : ConstVisitor {  
2      void accept(Definition obj) {  
3          super.accept(obj);  
4          this.definitionCnt++;  
5      }  
6  }
```

```
1  InlineFragment:
2      TDS: [on_, name#tc, Directives#dirs, SelectionSet#ss]
3      TS: [on_, name#tc, SelectionSet#ss]
4      DS: [Directives#dirs, SelectionSet#ss]
5      S: [SelectionSet#ss]
```

## Parser Example 2 1/2

```
1  InlineFragment parseInlineFragmentImpl() {
2      if(this.lex.front.type == TokenType.on_) {
3          this.lex.popFront();
4          if(this.lex.front.type == TokenType.name) {
5              Token tc = this.lex.front;
6              this.lex.popFront();
7              if(this.firstDirectives()) {
8                  Directives dirs = this.parseDirectives();
9                  if(this.firstSelectionSet()) {
10                     SelectionSet ss = this.parseSelectionSet();
11
12                     return new InlineFragment(InlineFragmentEnum.TDS, tc, dirs, ss);
13                 }
14                 throw new ParseException(["lcurly"]);
15             } else if(this.firstSelectionSet()) {
16                 SelectionSet ss = this.parseSelectionSet();
17
18                 return new InlineFragment(InlineFragmentEnum.TS, tc, ss);
19             }
```

## Parser Example 2 2/2

```
1      throw new ParseException(["at -> Directive","lcurly"]);
2  }
3      throw new ParseException(["name"]);
4  } else if(this.firstDirectives()) {
5      Directives dirs = this.parseDirectives();
6      if(this.firstSelectionSet()) {
7          SelectionSet ss = this.parseSelectionSet();
8
9          return new InlineFragment(InlineFragmentEnum.DS, dirs, ss);
10     }
11     throw new ParseException(["lcurly"]);
12 } else if(this.firstSelectionSet()) {
13     SelectionSet ss = this.parseSelectionSet();
14     return new InlineFragment(InlineFragmentEnum.S, ss);
15 }
16 throw new ParseException(["on_","at -> Directive","lcurly"]);
17 }
```



## Data-oriented Design (DoD)

---

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

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

	time	size
INST	$\approx$ 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

- Its called Abstract Syntax Tree not Abstract Syntax Array

- 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.



- 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

```
1  struct OperationDefinition {  
2      uint vdIdx;  
3      uint otIdx;  
4      uint dIdx;  
5      uint ssIdx;  
6      Token name;  
7      OperationDefinitionEnum ruleSelection;
```

## Parser Array 1/3

```
1  struct Parser {
2      Document[] documents;
3      Definitions[] definitionss;
4      Definition[] definitions;
5      OperationDefinition[] operationDefinitions;
6      SelectionSet[] selectionSets;
7      OperationType[] operationTypes;
8      Selections[] selectionss;
9      Selection[] selections;
10     FragmentSpread[] fragmentSpreads;
11     InlineFragment[] inlineFragments;
12     Field[] fields;
13     FieldName[] fieldNames;
14     Arguments[] argumentss;
15     ArgumentList[] argumentLists;
16     Argument[] arguments;
```

## Parser Array 2/3

```
1  uint parseOperationDefinitionImpl() {
2      string[] subRules;
3      if(this.firstSelectionSet()) {
4          uint ss = this.parseSelectionSet();
5
6          this.operationDefinitions ~= OperationDefinition.ConstructSelSet(ss);
7          return cast(uint)(this.operationDefinitions.length - 1);
8
9      } else if(this.firstOperationType()) {
10         uint ot = this.parseOperationType();
11         if(this.lex.front.type == TokenType.name) {
12             Token name = this.lex.front;
13             this.lex.popFront();
14             if(this.firstVariableDefinitions()) {
15                 uint vd = this.parseVariableDefinitions();
```

## Parser Array 3/3

```
1      uint vd = this.parseVariableDefinitions();
2      if(this.firstDirectives()) {
3          uint d = this.parseDirectives();
4          if(this.firstSelectionSet()) {
5              uint ss = this.parseSelectionSet();
6
7              this.operationDefinitions ~= OperationDefinition.ConstructOT_N_VD(ot,
name, vd, d, ss);
8              return cast(uint)(this.operationDefinitions.length - 1);
9
10         }
```

## Visitor Array 1/3

```
1 void accept(ref OperationDefinition obj) {
2     enter(obj);
3     final switch(obj.ruleSelection) {
4         case OperationDefinitionEnum.SelSet:
5             this.accept(this.parser.selectionSets[obj.ssIdx]);
6             break;
7         case OperationDefinitionEnum.OT_N_VD:
8             this.accept(this.parser.operationTypes[obj.otIdx]);
9             obj.name.visit(this);
10            this.accept(this.parser.variableDefinitionss[obj.vdIdx]);
11            this.accept(this.parser.directiveess[obj.dIdx]);
12            this.accept(this.parser.selectionSets[obj.ssIdx]);
13            break;
14        case OperationDefinitionEnum.OT_N_V:
15            this.accept(this.parser.operationTypes[obj.otIdx]);
16            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

## Structured Ranting

---



# AST Re-Structuring

```
1  OperationDefinition:
2      SelSet: [SelectionSet#ss]
3      OT_N_VD: [OperationType#ot, name#name, VariableDefinitions#vd, Directives#d,
4                SelectionSet#ss]
5      OT_N_V: [OperationType#ot, name#name, VariableDefinitions#vd, SelectionSet#ss]
6      OT_N_D: [OperationType#ot, name#name, Directives#d, SelectionSet#ss]
7      OT_N: [OperationType#ot, name#name, SelectionSet#ss]
8      OT_VD: [OperationType#ot, VariableDefinitions#vd, Directives#d, SelectionSet#
9              ss]
10     OT_V: [OperationType#ot, VariableDefinitions#vd, SelectionSet#ss]
11     OT_D: [OperationType#ot, Directives#d, SelectionSet#ss]
12     OT: [OperationType#ot, SelectionSet#ss]
```

# AST Re-Structuring

```
1  struct OperationDefinitionEnumFirst {
2      OperationDefinitionEnum ruleSelection : 4;
3      uint vdIdx : 28;
4  }
5
6  struct OperationDefinition {
7      OperationDefinitionEnumFirst vdIdx;
8      uint otIdx;
9      uint dIdx;
10     uint ssIdx;
11     uint name;
12 }
```

## Reading/Writing AST on Disk 1/2

```
1  void toDisk(ref File file) {
2      static foreach(mem; __traits(allMembers, Parser)) {{
3          alias T = typeof(__traits(getMember, Parser, mem));
4          static if(isArray!(T)) {
5              file.write(cast(uint) __traits(getMember, this, mem).length);
6              file.rawWrite(__traits(getMember, this, mem));
7          }
8      }}
9  }
```

## Reading/Writing AST on Disk 2/2

```
1  void fromDisk(ref File file) {
2      static foreach(mem; __traits(allMembers, Parser)) {{
3          alias T = typeof(__traits(getMember, Parser, mem));
4          static if(isArray!(T)) {
5              ubyte[4] lenA;
6              file.rawRead(lenA[]);
7              uint len = *(cast(uint*)lenA.ptr);
8              T[] arr = new T[len];
9              file.rawRead(arr);
10             __traits(getMember, this, mem) = arr;
11         }
12     }}
13 }
```

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

# TokenType

```
1  enum TokenType {
2      undefined
3      , exclamation
4      , dollar
5      , lparen
6      ...
7  }
8
9  struct Token {
10     string value;
11     uint line;
12     uint column;
13     TokenType type;
14 }
```

# TokenType

```
1  struct Token {
2      TokenType type : 7;
3      uint valueOrIndex : 25;
4  }
5
6  struct TokenPos {
7      uint line;
8      uint column;
9  }
10
11 struct Lexer {
12     int[] ints;
13     float[] floats;
14     double[] doubles;
15     TokenPos[16] positions;
16 }
```

# Lexer and Tokens

```
1  mutation MutateCreatePerson($legalName: LegalNameIn!  
2      , $knownAsName: KnownAsNameIn!  
3      , $privateContact: PrivateContactIn!  
4      , $activeAfter: DateTime  
5      , $includedInHeadcount: Boolean!) {  
6      createPerson(legalName: $legalName  
7          , knownAsName: $knownAsName  
8          , privateContact: $privateContact  
9          , activeAfter: $activeAfter  
10         , includedInHeadcount: $includedInHeadcount) {  
11          id  
12      }  
13 }
```

That graphql only contains 20 strings that need storing



## String/Array Interling

```
1 struct SmallStringPtr {
2     uint idx;
3     uint length;
4 }
5 struct StringInterling {
6     string str;
7     SmallStringPtr[] index;
8     uint[string] map;
9
10    uint insert(string s) {
11        uint* alreadyInMap = s in this.map;
12        if(alreadyInMap != null) return *alreadyInMap;
13        SmallStringPtr ptr;
14        ptr.index = cast(uint)this.str.length;
15        ptr.length = cast(uint)s.length;
16        this.str ~= s;
17        uint ret = cast(uint)this.index.length;
18        this.index ~= ptr;
19        return ret;
```

# String/Array Interling

```
1 struct SmallStringPtr {
2     uint idx;
3     uint length;
4 }
5 struct StringInterling {
6     string str;
7     SmallStringPtr[] index;
8     uint[string] map;
9
10    uint insert(string s) {
11        uint* alreadyInMap = s in this.map;
12        if(alreadyInMap != null) return *alreadyInMap;
13        SmallStringPtr ptr;
14        ptr.index = cast(uint)this.str.length;
15        ptr.length = cast(uint)s.length;
16        this.str ~= s;
17        uint ret = cast(uint)this.index.length;
18        this.index ~= ptr;
19        return ret;
```

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

Coming to an End

---

- 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

---