Лабораторная работа № 3.3 «Семантический анализ»

18 июня 2025 г.

Александр Старовойтов, ИУ9-61Б

Цель работы

Целью данной работы является получение навыков выполнения семантического анализа.

Индивидуальный вариант

Семантический анализ для варианта ЛР2.2

Определения структур, объединений и перечислений языка Си. В инициализаторах перечислений допустимы знаки операций +, -, *, /, sizeof, операндами могут служить имена перечислимых значений и целые числа.

Числовые константы могут быть только целочисленными и десятичными.

Проверки:

- Используемые идентификаторы должны быть определены выше по тексту.
- Теги структур, теги перечислений и теги объединений не должны повторяться.
- enum’ы определяют глобальные константы, они тоже не должны повторяться.
- В структурах и объединениях не могут встречаться одноимённые поля.

Результат:

- Программа должна выводить на экран значения всех констант.
- Для каждого типа должен вычисляться его объём. Считаем, что размеры целых чисел и перечислимых типов — 4 байта, вещественных чисел — 8 байт, размер указателя 4 байта. Считаем, что выравнивание не используется.

Реализация

```python
#!/usr/bin/env python3

import abc
import dataclasses
import enum
from dataclasses import dataclass
import parser_edsl as pe
from pprint import pprint
import sys
import itertools
import copy


@dataclass
class SemanticContext:
    definitions: list[str]
    is_top_level: bool
    position: pe.Position|None
    enum_variabels = dict()
    type_to_size = dict()


class SemanticError(pe.Error):
    def __init__(self, message, pos):
        self.pos = pos
        self.__message = message

    def message(self) -> str:
        return f'Ошибка {self.pos}: {self.__message}'


class DefinitionBase(abc.ABC):
    @abc.abstractmethod
    def check(self, ctx: SemanticContext):
        pass

    @abc.abstractmethod
    def type(self):
        return ""


# Definition -> Struct | Enum | Union
@dataclass
class Definition:
```

```
    data: DefinitionBase
    position: pe.Position
    definitions: list[str] = None
    enum_variabels: dict() = None

    @pe.ExAction
    def create(attrs, coords, _):
        attr = attrs[0]
        coord = coords[0].start
        return Definition(attr, coord)

    def check(self, ctx: SemanticContext):
        self.definitions = ctx.definitions
        self.enum_variabels = copy.deepcopy(ctx.enum_variabels)
        self.data.check(dataclasses.replace(ctx, position=self.position))


# NumType -> INT | DOUBLE | FLOAT | CHAR | SHORT | LONG
class Type(enum.Enum):
    INT = 'int'
    DOUBLE = 'double'
    FLOAT = 'float'
    CHAR = 'char'
    SHORT = 'short'
    LONG = 'long'


# Expr -> NUMBER | NAME | Expr + Expr | Expr - Expr | Expr * Expr | Expr / Expr | SIZEOF ( E
class Expr(abc.ABC):
    @abc.abstractmethod
    def calc(self, ctx: SemanticContext) -> int:
        pass


@dataclass
class Variable:
    name: str
    dimensions: list[Expr]
    position: pe.Position
    size: int|None = None

    @pe.ExAction
    def create(attrs, coords, _):
        coord = coords[0].start
        if len(attrs) == 3:
            return [Variable(*(attrs[:2]), coord)]+attrs[2]
```

```
        return [Variable(*attrs, coord)]

    def calc_size(self, ctx: SemanticContext, type_size: int):
        size = type_size
        for dimension in self.dimensions:
            size *= dimension.calc(ctx)
        self.size = size


@dataclass
class NumVariablesDefinition:
    typename: Type
    pointer_level: int
    variables: list[Variable]

    def check(self, ctx: SemanticContext):
        type_size = None
        if self.pointer_level > 0:
            type_size = 4
        else:
            match self.typename:
                case Type.INT:
                    type_size = 4
                case Type.DOUBLE:
                    type_size = 8
                case Type.FLOAT:
                    type_size = 8
                case Type.CHAR:
                    type_size = 1
                case Type.SHORT:
                    type_size = 2
                case Type.LONG:
                    type_size = 8
        for variable in self.variables:
            variable.calc_size(ctx, type_size)


def check(obj, ctx, type, size_strategy):
    prev, is_top_level, position = dataclasses.astuple(ctx)
    if obj.fields is not None and f'{type} {obj.name}' in prev:
        raise SemanticError(f'повторное определение {type} {obj.name}', position)

    def get_var_names(field):
        return [var.name for var in get_vars(field)]

    def get_var_positions(field):
```

```
            return [var.position for var in get_vars(field)]

        def get_vars(field):
            if isinstance(field, Definition):
                return [var for var in field.data.variables]
            return [var for var in field.variables]

        if obj.fields is not None:
            field_names = list(itertools.chain(*[get_var_names(field) for field in obj.fields]))
            field_positions = list(itertools.chain(*[get_var_positions(field) for field in obj.f
            for i, field in enumerate(field_names):
                if field in field_names[:i]:
                    raise SemanticError(f'поле с именем {field} уже объявлено', field_positions[

        if obj.pointer_level > 0:
            obj.size = 4
        elif obj.fields is not None:
            obj.size = 0
            for field in obj.fields:
                field.check(dataclasses.replace(ctx, is_top_level=False))
                obj.size = size_strategy(obj.size, size_strategy(*[var.size for var in get_vars(

            t = obj.type()
            if t and is_top_level:
                ctx.type_to_size[t] = obj.size
        else:
            if obj.type() not in ctx.type_to_size:
                raise SemanticError(f'{type} {obj.name} не определено', position)
            obj.size = ctx.type_to_size[obj.type()]


        for variable in obj.variables:
            variable.calc_size(ctx, obj.size)

        if not is_top_level and f'{type} {obj.name}' not in prev and obj.pointer_level == 0 and
            raise SemanticError(f'{type} {obj.name} не опредедено', position)

    # Struct -> STRUCT NameOpt StructFieldsOpt PointerOpt VariablesOpt ;
    @dataclass
    class Struct(DefinitionBase):
        name: str|None
        fields: list[Definition|NumVariablesDefinition]|None
        pointer_level: int
        variables: list[Variable]
        size: int|None = None
```

```
    def check(self, ctx: SemanticContext):
        check(self, ctx, 'struct', lambda *args: sum(args))

    def type(self):
        if not self.name:
            return ''
        return f'struct {self.name}'


@dataclass
class ExprNumber(Expr):
    number: int

    def calc(self, ctx: SemanticContext) -> int:
        return self.number


@dataclass
class ExprName(Expr):
    name: str
    position: pe.Position

    @pe.ExAction
    def create(attrs, coords, _):
        coord = coords[0].start
        return ExprName(*attrs, coord)

    def calc(self, ctx: SemanticContext) -> int:
        return ctx.enum_variabels[self.name][0]


@dataclass
class BinOpExpr(Expr):
    lhs: Expr
    op: str
    rhs: Expr

    def calc(self, ctx: SemanticContext) -> int:
        lhs = self.lhs.calc(ctx)
        rhs = self.rhs.calc(ctx)
        match self.op:
            case "+":
                return lhs + rhs
            case "-":
                return lhs - rhs
```

```
            case "*":
                return lhs * rhs
            case "/":
                return lhs // rhs


@dataclass
class UnOpExpr(Expr):
    op: str
    expr: Expr

    def calc(self, ctx: SemanticContext) -> int:
        match self.op:
            case "+":
                inner = self.expr.calc(ctx)
                return inner
            case "-":
                inner = self.expr.calc(ctx)
                return -inner
            case "sizeof":
                match self.expr:
                    case Type.INT:
                        return 4
                    case Type.DOUBLE:
                        return 8
                    case Type.FLOAT:
                        return 8
                    case Type.CHAR:
                        return 1
                    case Type.SHORT:
                        return 2
                    case Type.LONG:
                        return 8
                    case ExprNumber(e):
                        return e
                    case TypeStruct() | TypeUnion():
                        if self.expr.name in ctx.type_to_size:
                            return ctx.type_to_size[self.expr.name]
                        raise SemanticError(f'{self.expr.name} не определено', self.expr.pos

@dataclass
class TypeUnion:
    name: str
    position: pe.Position

    @pe.ExAction
```

```
    def create(attrs, coords, _):
        coord = coords[0].start
        return TypeUnion(f'union {attrs[0]}', coord)


class TypeStruct:
    name: str
    position: pe.Position

    @pe.ExAction
    def create(attrs, coords, _):
        coord = coords[0].start
        return TypeUnion(f'struct {attrs[0]}', coord)


# EnumField -> NAME EnumFieldRhsOpt
@dataclass
class EnumField:
    name: str
    rhs: Expr|None
    position: pe.Position
    value: int|None = None

    @pe.ExAction
    def create(attrs, coords, _):
        coord = coords[0].start
        if len(attrs) == 1:
            return EnumField(attrs[0], None, coord)
        return EnumField(*attrs, coord)

    def check(self, ctx: SemanticContext, index: int):
        if self.name in ctx.enum_variabels:
            raise SemanticError(f'{self.name} объявлено повторно', self.position)

        self.value = self.rhs.calc(ctx) if self.rhs is not None else index
        ctx.enum_variabels[self.name] = (self.value, self.position)



# Enum -> ENUM NameOpt EnumFieldsOpt PointerOpt VariablesOpt ;
@dataclass
class Enum(DefinitionBase):
    name: str|None
    fields: list[EnumField]
    pointer_level: int
    variables: list[Variable]
```

```
    size: int = 4

    def check(self, ctx: SemanticContext):
        prev, _, position = dataclasses.astuple(ctx)
        if self.fields is not None and f'enum {self.name}' in prev:
            raise SemanticError(f'enum {self.name} объявлено повторно', position)
        for i, field in enumerate(self.fields):
            field.check(ctx, i)

        for variable in self.variables:
            variable.size = 4

    def type(self):
        if not self.name:
            return ''
        return f'struct {self.name}'


# Union -> UNION NameOpt UnionFieldsOpt PointerOpt VariablesOpt ;
@dataclass
class Union(DefinitionBase):
    name: str|None
    fields: list[Definition|NumVariablesDefinition]|None
    pointer_level: int
    variables: list[Variable]
    size: int|None = None

    def check(self, ctx: SemanticContext):
        check(self, ctx, 'union', max)

    def type(self):
        if not self.name:
            return ''
        return f'union {self.name}'


# Program -> Definition Program | Definition
@dataclass
class Program:
    definitions: list[Definition]

    def check(self):
        defined = list()
        for i, d in enumerate(self.definitions):
            d.check(SemanticContext(defined, True, None))
            type = d.data.type()
```

```
            if type != '':
                defined.append(d.data.type())


INTEGER = pe.Terminal('INTEGER', '[0-9]+', int, priority=7)
VARNAME = pe.Terminal('VARNAME', '[A-Za-z][A-Za-z0-9_]*', str)

def make_keyword(image):
    return pe.Terminal(image, image, lambda name: None, priority=10)

KW_INT, KW_DOUBLE, KW_FLOAT, KW_CHAR, KW_SHORT, KW_LONG = \
    map(make_keyword, 'int double float char short long'.split())

KW_SIZEOF, KW_ENUM, KW_UNION, KW_STRUCT = \
    map(make_keyword, 'sizeof enum union struct'.split())

NProgram, NDefinition, NDefinitions, NStruct, NEnum, NUnion = \
    map(pe.NonTerminal, 'Program Definition Definitions Struct Enum Union'.split())

NDefinitionOrVariable, NNumVar, NNumType, NPointerOpt, NVariables = \
    map(pe.NonTerminal, 'DefinitionOrVariable NumVar NumType PointerOpt Variables'.split())

NNameOpt, NStructFieldsOpt, NVariablesOpt, NStructFields = \
    map(pe.NonTerminal, 'NameOpt StructFieldsOpt VariablesOpt StructFields'.split())

NDefinitionsOrVariables, NVariablesTail, NEnumFieldsOpt, NSizeOf  = \
    map(pe.NonTerminal, 'DefinitionsOrVariables VariablesTail, EnumFieldsOpt SizeOf'.split()

NEnumFields, NEnumField, NEnumFieldsTail, NCommaOpt, NDimensions  = \
    map(pe.NonTerminal, 'EnumFields EnumField EnumFieldsTail CommaOpt Dimensions'.split())

NEnumFieldRhsOpt, NExpr, NUnionFieldsOpt, NUnionFields = \
    map(pe.NonTerminal, 'EnumFieldRhsOpt Expr UnionFieldsOpt UnionFields'.split())

NEnumOther, NEnumOtherOther, NEnumFieldsRest, NEnumFieldsBody = \
    map(pe.NonTerminal, 'EnumOther EnumOtherOther EnumFieldsRest EnumFieldsBody'.split())

NFactor, NTerm, NAddOp, NMulOp = \
    map(pe.NonTerminal, 'Factor Term AddOp MulOp'.split())

# Program -> Definition Program | Definition
NProgram |= NDefinitions, Program
NDefinitions |= NDefinition, NDefinitions, lambda d, p: [d]+p
NDefinitions |= lambda: []

# Definition -> Struct | Enum | Union
```

```
NDefinition |= NStruct, Definition.create
NDefinition |= NEnum, Definition.create
NDefinition |= NUnion, Definition.create

# DefinitionOrVariable -> Definition | NumVar
NDefinitionOrVariable |= NDefinition
NDefinitionOrVariable |= NNumVar

# NumVar -> NumType PointerOpt Variables ;
NNumVar |= NNumType, NPointerOpt, NVariables, ';', NumVariablesDefinition

# PointerOpt -> * PointerOpt | ε
NPointerOpt |= '*', NPointerOpt, lambda x: x+1
NPointerOpt |= lambda: 0

# NumType -> INT | DOUBLE | FLOAT | CHAR | SHORT | LONG
NNumType |= KW_INT, lambda: Type.INT
NNumType |= KW_DOUBLE, lambda: Type.DOUBLE
NNumType |= KW_CHAR, lambda: Type.CHAR
NNumType |= KW_SHORT, lambda: Type.SHORT
NNumType |= KW_LONG, lambda: Type.LONG

# Struct -> STRUCT NameOpt StructFieldsOpt PointerOpt VariablesOpt ;
NStruct |= KW_STRUCT, NNameOpt, NStructFields, NPointerOpt, NVariablesOpt, ';', Struct
NStruct |= KW_STRUCT, VARNAME, NStructFieldsOpt, NPointerOpt, NVariablesOpt, ';', Struct

# NameOpt -> NAME | ε
NNameOpt |= VARNAME
NNameOpt |= lambda: ''

# VariablesOpt -> Variables | ε
NVariablesOpt |= NVariables
NVariablesOpt |= lambda: []

# StructFieldsOpt -> StructFields | ε
NStructFieldsOpt |= NStructFields
NStructFieldsOpt |= lambda: None

# StructFields -> { DefinitionsOrVariables }
NStructFields |= '{', NDefinitionsOrVariables, '}'

# DefinitionsOrVariables -> DefinitionOrVariable DefinitionsOrVariables | ε
NDefinitionsOrVariables |= NDefinitionOrVariable, NDefinitionsOrVariables, lambda d, arr: [d
NDefinitionsOrVariables |= lambda: []

# Variables -> NAME , Variables | Variables
```

```
NVariables |= VARNAME, NDimensions, ',', NVariables, Variable.create
NVariables |= VARNAME, NDimensions, Variable.create

NDimensions |= '[', NExpr, ']', NDimensions, lambda expr, other: [expr]+other
NDimensions |= lambda: []


NEnum |= KW_ENUM, VARNAME, NEnumOther, lambda name, other: Enum(name, *other)
NEnum |= KW_ENUM, NEnumOther, lambda other: Enum('', *other)

NEnumOther |= NEnumFields, NEnumOtherOther, lambda fields, other: [fields]+other
NEnumOther |= NEnumOtherOther, lambda other: [[]]+other

NEnumOtherOther |= NPointerOpt, NVariables, ';', lambda p, v: [p, v]
NEnumOtherOther |= ';', lambda: [0, []]

NEnumFields |= '{', NEnumFieldsBody

NEnumFieldsBody |= '}', lambda: []
NEnumFieldsBody |= NEnumField, NEnumFieldsRest, lambda l, r: [l]+r
NEnumFieldsRest |= ',', NEnumField, NEnumFieldsRest, lambda l, r: [l]+r
NEnumFieldsRest |= ',', '}', lambda: []
NEnumFieldsRest |= '}', lambda: []

# EnumFieldsTail -> , EnumField EnumFieldsTail | ε
NEnumFieldsTail |= ',', NEnumField, NEnumFieldsTail, lambda f, t: [f]+t
NEnumFieldsTail |= lambda: []

# EnumField -> NAME EnumFieldRhsOpt
NEnumField |= VARNAME, '=', NExpr, EnumField.create
NEnumField |= VARNAME, EnumField.create

# Expr -> NUMBER | NAME | Expr + Expr | Expr - Expr | Expr * Expr | Expr / Expr | SIZEOF ( V
NExpr |= NTerm
NExpr |= '+', NTerm, lambda t: UnOpExpr('+', t)
NExpr |= '-', NTerm, lambda t: UnOpExpr('-', t)
NExpr |= NExpr, NAddOp, NTerm, BinOpExpr
NTerm |= NFactor
NTerm |= NTerm, NMulOp, NFactor, BinOpExpr
NMulOp |= '*', lambda: '*'
NMulOp |= '/', lambda: '/'
NAddOp |= '+', lambda: '+'
NAddOp |= '-', lambda: '-'
NFactor |= INTEGER, lambda v: ExprNumber(int(v))
NFactor |= '(', NExpr, ')'
NFactor |= KW_SIZEOF, '(', NSizeOf, ')', lambda inner: UnOpExpr('sizeof', inner)
```

```
NFactor |= VARNAME, ExprName.create
NSizeOf |= KW_UNION, VARNAME, TypeUnion.create
NSizeOf |= KW_STRUCT, VARNAME, TypeStruct.create
NSizeOf |= NNumType

# Union -> UNION NameOpt UnionFieldsOpt PointerOpt VariablesOpt ;
NUnion |= KW_UNION, VARNAME, NUnionFieldsOpt, NPointerOpt, NVariablesOpt, ';', Union
NUnion |= KW_UNION, NNameOpt, NUnionFields, NPointerOpt, NVariablesOpt, ';', Union

# UnionFieldsOpt -> UnionFields | ε
NUnionFieldsOpt |= NUnionFields
NUnionFieldsOpt |= lambda: []

# UnionFields -> { DefinitionsOrVariables }
NUnionFields |= '{', NDefinitionsOrVariables, '}'

p = pe.Parser(NProgram)

p.add_skipped_domain('\\s')
p.add_skipped_domain(r"(?:\/\/.*)|(?:\/\*(?:.|\n)*?\*\/)")

for filename in sys.argv[1:]:
    with open(filename) as f:
        try:
            tree = p.parse_earley(f.read())
            tree.check()
            pprint(tree)
        except SemanticError as e:
            print(e.message())
        except Exception as e:
            raise e
        else:
            print('Программа корректна')
```

Тестирование

Входные данные

```
enum {
  BUFFER_SIZE = 100000,
  PAGE_SIZE = 4096,
  PAGES_FOR_BUFFER = (BUFFER_SIZE + PAGE_SIZE - 1) / PAGE_SIZE
};
```

Вывод на `stdout`

```
Program(definitions=[Definition(data=Enum(name='',
                                fields=[EnumField(name='BUFFER_SIZE',
                                                  rhs=ExprNumber(number=100000),
                                                  position=Position(offset=9,
                                                                    line=2,
                                                                    col=3),
                                                               value=100000),
                                                EnumField(name='PAGE_SIZE',
                                                  rhs=ExprNumber(number=4096),
                                                  position=Position(offset=33,
                                                                    line=3,
                                                                    col=3),
                                                                 value=4096),
                                                EnumField(name='PAGES_FOR_BUFFER',
                                                  rhs=BinOpExpr(lhs=BinOpExpr(
                                                  lhs=BinOpExpr(lhs=ExprName(name='BUFFER_SIZE',
                                                  position=Position(offset=73,
                                                                    line=4,
                                                                  col=23)),
                                                                    op='+',
                                                  rhs=ExprName(name='PAGE_SIZE',
                                                  position=Position(offset=87,
                                                                    line=4,
                                                                 col=37))),
                                                                    op='-',
                                                  rhs=ExprNumber(number=1)),
                                                                    op='/',
                                                           rhs=ExprName(name='PAGE_SIZE',
                                                     position=Position(offset=104,
                                                                    line=4,
                                                                 col=54))),
                                                  position=Position(offset=53,
                                                                    line=4,
                                                                    col=3),
                                                                 value=25)],
                                                    pointer_level=0,
                                                    variables=[],
                                                    size=4),
                                position=Position(offset=0, line=1, col=1),
                                         definitions=[],
                                         enum_variabels={})])
Программа корректна
```

Вывод

В рамках выполнения данной работы я получил навыки выполнения семантического анализа.