Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина: Методы трансляции

ОТЧЁТ
по лабораторной работе
на тему

Интерпретация исходного кода

Выполнил
Студент гр. 053501
Хиль В.М.

Проверил
Ассистент кафедры информатики
Гриценко Н.Ю.

Минск 2023

СОДЕРЖАНИЕ

1 ЦЕЛЬ РАБОТЫ

На основе результатов анализа лабораторных работ 1 – 4 выполнить интерпретацию программы.

2 КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Интерпретация кода – это процесс выполнения исходного кода на выбранном языке программирования путем его последовательного чтения и выполнения команд.

Простой интерпретатор анализирует и тут же выполняет (собственно интерпретация) программу покомандно или построчно по мере поступления её исходного кода на вход интерпретатора. Достоинством такого подхода является мгновенная реакция. Недостаток — такой интерпретатор обнаруживает ошибки в тексте программы только при попытке выполнения команды или строки с ошибкой.

3 ПРИМЕРЫ РАБОТЫ ИНТЕРПРЕТАТОРА

Возьмём программу нахождения факториала, которую рассмотрели в первой лабораторной работе (см. рисунок 1).

```
FUN fact(n)
    VAR result = 1
    IF n < 0 THEN
        PRINT("Error! Factorial of a negative number doesn't exist.")
    ELSE
        FOR i = 1 TO n+1 THEN
            VAR result = result * i
        END
    END
    RETURN result
END
```

Рисунок 1 – Пример программы

Примеры выполнения программы для разных входных чисел (5 и 10) представлены на рисунках 2 и 3.

```
Factorial for:
5
Equal:
120
```

Рисунок 2 – Пример выполнения программы для числа 5

```
Factorial for:
10
Equal:
3628800
```

Рисунок 3 – Пример выполнения программы для числа 10

Никакая программа не застрахована от шибок времени выполнения. Даже учитывая, что множество потенциальных проблем было выявлено на предыдущих стадиях до непосредственного исполнения, в любом случае могут случиться ситуации, в которых какие-то данные не соответствуют ожидаемым. В примере выше таким местом является выражение fact(n), используемое для получения пользовательского числа в переменную n. Данный код выполняется корректно, если пользователь передал в функцию

число. Тем не менее, ничего не мешает передать строку, не представимую в виде числа. В таком случае мы получим ошибку времени выполнения программы (см. рисунок 4).

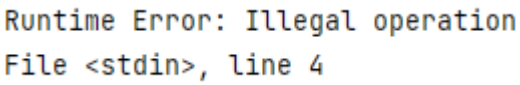

Рисунок 4 – Пример ошибки времени выполнения

ПРИЛОЖЕНИЕ А
(обязательное)
Листинг кода

```
class Interpreter:
    def visit_method(self, node, context):
        method_name = f'visit_{type(node).__name__}'
        method = getattr(self, method_name, self.no_visit_method)
        return method(node, context)

    def no_visit_method(self, node, context):
        raise Exception(f'No visit_{type(node).__name__} method defined')

    ###################################

    def visit_ListNode(self, node, context):
        res = RTResult()
        elements = []

        for element_node in node.nodes:
            elements.append(res.register(self.visit_method(element_node,
context)))

            if res.should_return():
                return res

        return res.success(

List(elements).set_context(context).set_position(node.start_position,
node.end_position)
        )

    def visit_BinaryOperationNode(self, node, context):
        res = RTResult()
        left = res.register(self.visit_method(node.left_node, context))

        if res.should_return():
            return res

        right = res.register(self.visit_method(node.right_node, context))

        if res.should_return():
            return res

        elif node.op_token.type == MUL:
            result, error = left.multed_by(right)

        elif node.op_token.type == DIV:
            result, error = left.dived_by(right)

        if node.op_token.type == PLUS:
            result, error = left.added_to(right)

        elif node.op_token.type == MINUS:
            result, error = left.subbed_by(right)

        elif node.op_token.type == POW:
            result, error = left.powed_by(right)

        elif node.op_token.type == EE:
```

```
            result, error = left.get_comparison_eq(right)

        elif node.op_token.type == NE:
            result, error = left.get_comparison_ne(right)

        elif node.op_token.type == GT:
            result, error = left.get_comparison_gt(right)

        elif node.op_token.type == GTE:
            result, error = left.get_comparison_gte(right)

        elif node.op_token.type == LT:
            result, error = left.get_comparison_lt(right)

        elif node.op_token.type == LTE:
            result, error = left.get_comparison_lte(right)

        elif node.op_token.matches(KEYWORD, 'AND'):
            result, error = left.anded_by(right)

        elif node.op_token.matches(KEYWORD, 'OR'):
            result, error = left.ored_by(right)

        if error:
            return res.failure(error)
        else:
            return res.success(result.set_position(node.start_position,
node.end_position))

    def visit_UnaryOperationNode(self, node, context):
        res = RTResult()
        number = res.register(self.visit_method(node.node, context))

        if res.should_return():
            return res

        error = None

        if node.op_token.type == MINUS:
            number, error = number.multed_by(Number(-1))

        elif node.op_token.matches(KEYWORD, 'NOT'):
            number, error = number.notted()

        if error:
            return res.failure(error)
        else:
            return res.success(number.set_position(node.start_position,
node.end_position))

    def visit_NumberNode(self, node, context):
        return RTResult().success(

Number(node.token.value).set_context(context).set_position(node.start_positio
n, node.end_position)
        )

    def visit_StringNode(self, node, context):
        return RTResult().success(

String(node.token.value).set_context(context).set_position(node.start_positio
n, node.end_position)
```

```
        )

    def visit_VarAccessNode(self, node, context):
        res = RTResult()
        var_name = node.var_name_token.value
        value = context.symbol_table.get(var_name)

        if not value:
            return res.failure(RTError(
                node.start_position, node.end_position,
                f"'{var_name}' is not defined",
                context
            ))

        value = value.copy().set_position(node.start_position,
node.end_position).set_context(context)

        return res.success(value)

    def visit_VarAssignNode(self, node, context):
        res = RTResult()
        var_name = node.var_name_token.value
        value = res.register(self.visit_method(node.value_node, context))

        if res.should_return():
            return res

        context.symbol_table.set(var_name, value)

        return res.success(value)

    def visit_IfNode(self, node, context):
        res = RTResult()

        for condition, expr, should_return_null in node.cases:
            condition_value = res.register(self.visit_method(condition,
context))

            if res.should_return():
                return res

            if condition_value.is_true():
                expr_value = res.register(self.visit_method(expr, context))

                if res.should_return():
                    return res

                return res.success(Number.null if should_return_null else
expr_value)

        if node.else_case:
            expr, should_return_null = node.else_case
            expr_value = res.register(self.visit_method(expr, context))

            if res.should_return():
                return res

            return res.success(Number.null if should_return_null else
expr_value)

        return res.success(Number.null)
```

```
    def visit_WhileNode(self, node, context):
        res = RTResult()
        elements = []

        while True:
            condition = res.register(self.visit_method(node.condition_node,
context))

            if res.should_return():
                return res

            if not condition.is_true():
                break

            value = res.register(self.visit_method(node.body_node, context))

            if res.should_return() and res.loop_should_continue == False and
res.loop_should_break == False:
                return res

            if res.loop_should_continue:
                continue

            if res.loop_should_break:
                break

            elements.append(value)

        return res.success(
            Number.null if node.should_return_null else

List(elements).set_context(context).set_position(node.start_position,
node.end_position)
        )

    def visit_ForNode(self, node, context):
        res = RTResult()
        elements = []

        start_value = res.register(self.visit_method(node.start_value_node,
context))

        if res.should_return():
            return res

        end_value = res.register(self.visit_method(node.end_value_node,
context))

        if res.should_return():
            return res

        if node.step_value_node:
            step_value = res.register(self.visit_method(node.step_value_node,
context))

            if res.should_return():
                return res
        else:
            step_value = Number(1)

        i = start_value.value
```

```
        if step_value.value >= 0:
            condition = lambda: i < end_value.value
        else:
            condition = lambda: i > end_value.value

        while condition():
            context.symbol_table.set(node.var_name_token.value, Number(i))
            i += step_value.value

            value = res.register(self.visit_method(node.body_node, context))
            if res.should_return() and res.loop_should_continue == False and
res.loop_should_break == False:
                return res

            if res.loop_should_continue:
                continue

            if res.loop_should_break:
                break

            elements.append(value)

        return res.success(
            Number.null if node.should_return_null else

List(elements).set_context(context).set_position(node.start_position,
node.end_position)
        )

    def visit_ContinueNode(self, node, context):
        return RTResult().success_continue()

    def visit_BreakNode(self, node, context):
        return RTResult().success_break()

    def visit_FuncDefinitionNode(self, node, context):
        res = RTResult()

        func_name = node.var_name_token.value if node.var_name_token else
None
        body_node = node.body_node
        arguments_names = [arguments_name.value for arguments_name in
node.arguments_name_tokens]
        func_value = Function(func_name, body_node, arguments_names,
node.should_auto_return).set_context(
            context).set_position(
            node.start_position, node.end_position)

        if node.var_name_token:
            context.symbol_table.set(func_name, func_value)

        return res.success(func_value)

    def visit_CallNode(self, node, context):
        res = RTResult()
        arguments = []

        value_to_call = res.register(self.visit_method(node.node_to_call,
context))

        if res.should_return():
            return res
```

```
        value_to_call =
value_to_call.copy().set_position(node.start_position, node.end_position)

        for arguments_node in node.arguments_nodes:
            arguments.append(res.register(self.visit_method(arguments_node,
context)))

            if res.should_return():
                return res

        return_value = res.register(value_to_call.execute(arguments))

        if res.should_return():
            return res

        return_value = return_value.copy().set_position(node.start_position,
node.end_position).set_context(context)

        return res.success(return_value)

    def visit_ReturnNode(self, node, context):
        res = RTResult()

        if node.node_to_return:
            value = res.register(self.visit_method(node.node_to_return,
context))

            if res.should_return():
                return res

        else:
            value = Number.null

        return res.success_return(value)
```