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 3](#_Toc132146010)

[2 Краткие теоретические сведения 4](#_Toc132146011)

[3 Примеры работы интерпретатора 5](#_Toc132146012)

[Приложение А (обязательное) Листинг кода 7](#_Toc132146013)

**1 ЦЕЛЬ РАБОТЫ**

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

1. **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.

1. **ПРИМЕРЫ РАБОТЫ ИНТЕРПРЕТАТОРА**

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

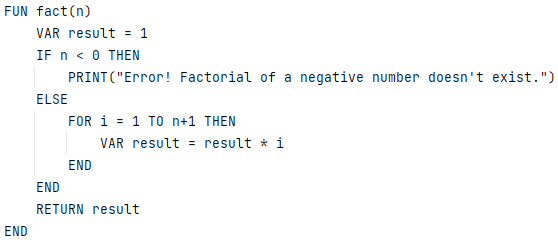


Рисунок 1 – Пример программы

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

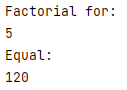


Рисунок 2 – Пример выполнения программы для числа 5

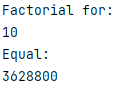


Рисунок 3 – Пример выполнения программы для числа 10

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\_position, node.end\_position)  
 )  
  
 def visit\_StringNode(self, node, context):  
 return RTResult().success(  
 String(node.token.value).set\_context(context).set\_position(node.start\_position, 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)