Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина Методы трансляции

|  |  |
| --- | --- |
|  |  |

ОТЧЕТ

к лабораторной работе № 5

на тему

**ИНТЕРПРЕТАЦИЯ ИСХОДНОГО КОДА**

Выполнил             В. П. Бычко

Проверил                          Н. Ю. Гриценко

Минск 2024

СОДЕРЖАНИЕ

[1 Постановка задачи 3](#_Toc157960226)

[2 Краткие теоритические сведения 4](#_Toc157960235)

[3 Результаты выполнения лабораторной работы 5](#_Toc157960235)

[Выводы 7](#_Toc157960236)

[Список использованных источников 8](#_Toc157960237)

[Приложение А (обязательное) Листинг програмного кода 9](#_Toc157960238)

## 1 ПОСТАНОВКА ЗАДАЧИ

Целью выполнения лабораторной работы является разработка интерпретатора подмножества языка программирования C. Выполнение интерпретация исходного кода.

# 2 КРАТКИЕ ТЕОРИТИЧЕСКИЕ СВЕДЕНИЯ

Интерпретатор – это программа, которая выполняет код, написанный на языке программирования, путем последовательного чтения и выполнения инструкций из исходного кода. Когда вы пишете программу на языке программирования, интерпретатор анализирует каждую строку кода и немедленно выполняет соответствующие действия. [1]

В отличие от компилятора, который преобразует весь исходный код в машинный код до его выполнения, интерпретатор работает пошагово, переводя и выполняя инструкции по мере необходимости. Это означает, что интерпретатор может начать выполнение программы даже если она не полностью завершена или содержит ошибки.

Интерпретаторы широко используются в различных языках программирования, таких как Python, JavaScript, Ruby, PHP и других. Они обеспечивают гибкость и удобство при разработке программ, позволяя разработчикам быстро тестировать и экспериментировать с кодом.

Интерпретаторы также обеспечивают возможность взаимодействия с пользователем в режиме реального времени, например, через интерактивную консоль или среду разработки. Это делает процесс отладки и тестирования кода более удобным и эффективным. [2]

# 3 РЕЗУЛЬТАТЫ ВЫПОЛНЕНИЯ ЛАБОРАТОРНОЙ РАБОТЫ

Во время выполнения лабораторной работы был реализовал семантический анализатор языка С. Скриншот результатов проверки ветвления представлен на рисунке 3.1.

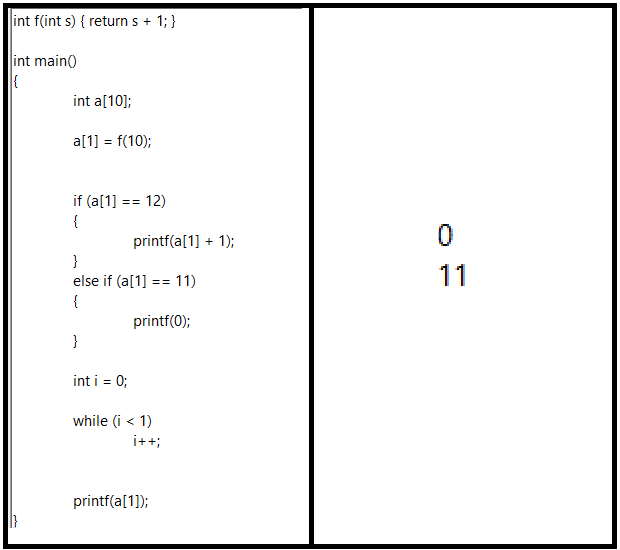


Рисунок 3.1 – Заданный код и результат проверки ветвления

Вторая проверка – проверка работы указателей 3.2. Скриншот результатов проверки представлен на рисунке 3.2.

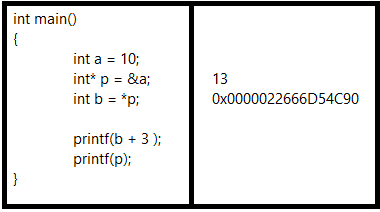


Рисунок 3.2 – Заданный код и результат проверки указателей

Третья проверка – проверка работы со структурами. Скриншот результатов проверки представлен на рисунке 3.3.

## 

Рисунок 3.3 – Заданный код и результат проверки структур

## ВЫВОДЫ

В ходе выполнения лабораторной работы по написанию интерпретатора были изучены основные принципы интерпретации необходимые для удачного выполнения скрипта. Был реализован семантический анализатор, способный проверять правильность построенное синтаксическое дерево на соответствие семантике языка Си.

## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Основы интерпритацииа [Электронный ресурс]. – Режим доступа:

https://blog.skillfactory.ru/glossary/interpretator/– Дата доступа: 18.03.2024

1. Введение в теорию компиляторов [Электронный ресурс]. – Режим доступа:https://uchet-jkh.ru/i/razlicie-mezdu-kompilyaciei-i-interpretaciei-v-programmirovanii/– Дата доступа: 10.04.2024

**ПРИЛОЖЕНИЕ А**

**(обязательное)**

**Листинг программного кода**

public static partial class Parser

{

static List<dynamic> objects = [];

public unsafe static void Interpret(Node root, StringWriter console)

{

GCLatencyMode oldMode = GCSettings.LatencyMode;

RuntimeHelpers.PrepareConstrainedRegions();

dynamic? main = null;

try

{

GCSettings.LatencyMode = GCLatencyMode.LowLatency;

Scope mainScope = new Scope();

object @return = new object();

Interpret(root, console, mainScope, ref @return);

if (main is null)

{

throw new InterpretationException("No main.");

}

else

{

main(Array.Empty<object>());

}

}

catch (Exception e)

{

throw new InterpretationException(e.Message, e);

}

finally

{

GCSettings.LatencyMode = oldMode;

}

void Interpret(Node root, StringWriter console, Scope scope, ref object? @return)

{

if (root is OperatorNode operatorNode)

{

string @operator = operatorNode.Operator;

if (false)

{

}

if (@operator == "Function declaration")

{

Node type = operatorNode.Children.ToArray()[0];

Token funcToken = ((ValueNode)operatorNode.Children.ToArray()[1]).Token;

IEnumerable<Node> @params = ((OperatorNode)operatorNode.Children.ToArray()[2]).Children;

Dictionary<int, int> paramsIndexes = @params.Select((p, index) =>

{

int varId = ((ValueNode)((OperatorNode)p).Children.ToArray()[1]).Token.Id;

return KeyValuePair.Create(index, varId);

})

.ToDictionary();

Node body = operatorNode.Children.ToArray()[3];

var allParamsScope = new Scope(parentScope: scope);

foreach (Node param in @params)

{

Interpret(param, console, allParamsScope, ref @return);

}

Delegate func = (params dynamic[] values) =>

{

object? @return = new object();

for (int i = 0; i < values.Length; i++)

{

scope.Variables[paramsIndexes[i]] = values[i];

}

Interpret(body, console, scope, ref @return);

return @return;

};

dynamic funcValue = new FunctionValue(func);

scope.Variables[funcToken.Id] = funcValue;

if (funcToken.Value == "main")

main = funcValue;

}

else if (@operator == "Function prototype")

{ }

else if (@operator == "Struct declaration")

{ }

else if (@operator == "Variable declaration")

{

Node typeNode = operatorNode.Children.ToArray()[0];

Token token = ((ValueNode)operatorNode.Children.ToArray()[1]).Token;

TypeInfo typeInfo;

if (typeNode is ValueNode valueNode)

{

typeInfo = new TypeInfo(typesParts: [valueNode.Token.Value], structs: []);

}

else if (typeNode is TypesNode complexTypeNode)

{

typeInfo = complexTypeNode.TypeInfo;

}

else

{

throw new UnexpectedException();

}

List<int> arrayDeclarations = operatorNode.Children.ToArray()[2..].Select(a =>

{

if (a is OperatorNode o && o.Operator.StartsWith("Array"))

{

if (o.Children.Any())

{

return int.Parse(((ValueNode)o.Children.First()).Token.Value);

}

else

{

return 100;

}

}

else

{

throw new UnexpectedException();

}

})

.ToList();

dynamic variable = CreateVariable(typeInfo, arrays: arrayDeclarations);

int varialeId = token.Id;

scope.Variables[varialeId] = variable;

}

else if (@operator == "Variable initialization")

{

var initValueNode = operatorNode.Children.ToArray()[1];

var declarationNode = (OperatorNode)operatorNode.Children.ToArray()[0];

Token token = ((ValueNode)declarationNode.Children.ToArray()[1]).Token;

Interpret(declarationNode, console, scope, ref @return);

dynamic initValue = GetRValue(initValueNode, scope);

scope.Variables[token.Id] = initValue;

}

else if (@operator is "If else if statements" or "If else if else statements")

{

var statements = operatorNode.Children.ToArray();

List<(Node?, Node)> conditionsBlocks = statements.Select(s =>

{

Node[] conditionWithBlock = ((OperatorNode)s).Children.ToArray();

if (conditionWithBlock.Length == 1) // else case

{

return (null, conditionWithBlock[0])!;

}

else

{

return (conditionWithBlock[0], conditionWithBlock[1]);

}

})

.ToList()!;