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 №1

на тему

**ОПРЕДЕЛЕНИЕ МОДЕЛИ ЯЗЫКА. ВЫБОР ИНСТРУМЕНТАЛЬНОЙ ЯЗЫКОВОЙ СРЕДЫ**

Студент Т. П. Власенко

Преподаватель Н. Ю. Гриценко

Минск 2024

СОДЕРЖАНИЕ

[1 Цель работы 3](#_Toc158362079)

[2 Подмножество языка программирования 4](#_Toc158362080)

[3 Инструментальная языковая среда 10](#_Toc158362081)

[Заключение 11](#_Toc158362082)

[Список использованных источников 12](#_Toc158362083)

[Приложение А (обязательное) Текст программ 13](#_Toc158362084)

1. **ЦЕЛЬ РАБОТЫ**

Цель работы заключается в определении подмножества языка программирования (типы констант, переменных, операторов и функций). В подмножество языке необходимо включить все типы. Необходимо учесть все структуры, представленные в рассматриваемом языке. Учесть подключение библиотек (модулей) и так далее. Операторы цикла *(do...while*, *for)*, которые поддерживает язык. Условные операторы *(if...else, case)*, которые поддерживает язык*.* Определение инструментальной языковой среды, т.е. языка программирования и операционной системы для разработки включает: язык программирования c указанием версии, на котором ведётся разработка*,* операционная система (*Windows, Linux* и т.д.), в которой выполняется разработка, компьютер (*PC / Macintosh*). В отчете по лабораторной работе дается полное определение подмножества языка программирования, тексты 2-3-х программ, включающих все элементы этого подмножества. Приводится подробное описание инструментальной языковой среды.

1. **ПОДМНОЖЕСТВО ЯЗЫКА ПРОГРАММИРОВАНИЯ**

В языке программирования *Golang* существует множество типов данных, которые можно использовать для объявления переменных.

Логический тип представляет набор логических значений истинности, обозначаемых предварительно объявленными константами *true* и *false*. Предварительно объявленным логическим типом является *bool*.

Числовые типы:

1. *Int* – целые числа со знаком, размер которых зависит от платформы (обычно 32 или 64 бита).
2. *Int8*, *int16*, *int32*, *int64* – целые числа со знаком фиксированного размера (от 8 до 64 бит).
3. *Uint* – целые числа без знака, размер которых зависит от платформы (32 или 64 бита).
4. *Uint8*, *uint16*, *uint32*, *uint64* – целые числа без знака фиксированного размера (от 8 до 64 бит).
5. *Uintptr* – целое число без знака, достаточно большое для хранения неинтерпретированных битов значения указателя.
6. *Float32* – набор всех 32-разрядных чисел с плавающей запятой стандарта IEEE-754.
7. *Float64* – набор всех 64-разрядных чисел с плавающей запятой стандарта IEEE-754.
8. *Complex64* – множество всех комплексных чисел с *float*32 действительной и мнимой частями.
9. *Complex128* – множество всех комплексных чисел с *float*64 действительной и мнимой частями.
10. *Byte* – псевдоним для *uint8*.
11. *Rune* – псевдоним для *int32*, используется для представления символов *Unicode*.

Строковый тип – *string*. Строковый тип представляет набор строковых значений. Строковое значение – это (возможно, пустая) последовательность байтов. Количество байтов называется длиной строки и никогда не бывает отрицательным. Строки неизменяемы: после создания невозможно изменить содержимое строки.

Существуют логические константы, рунные константы, целочисленные константы, константы с плавающей запятой, комплексные константы и строковые константы. Рунные, целочисленные, с плавающей запятой и комплексные константы в совокупности называются числовыми константами.

В *Golang* представлены массивы (рисунок 1): *ArrayType* =[*ArrayLength*] *ElementType*. Массив ­– это пронумерованная последовательность элементов одного типа, называемого типом элемента. Количество элементов называется длиной массива и никогда не бывает отрицательным.

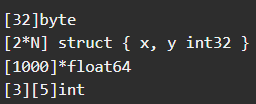


Рисунок 1 – Массивы в языке *Golang*

Помимо массивов, которые имеют фиксированный размер, в *Golang* представлены срезы (рисунок 2). Срез является дескриптором для непрерывного сегмента базового массива и предоставляет доступ к пронумерованной последовательности элементов из этого массива.

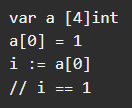


Рисунок 2 – Срезы в языке *Golang*

Карта – это неупорядоченная группа элементов одного типа, называемая типом элемента, индексируемая набором уникальных ключей другого типа, называемых типом ключа (рисунок 3). Значение неинициализированной карты равно *nil*.

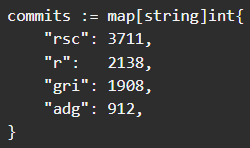


Рисунок 3 – Карты в языке *Golang*

Тип интерфейса определяет набор типов (рисунок 4). Переменная типа интерфейса может хранить значение любого типа, которое находится в наборе типов интерфейса. Говорят, что такой тип реализует интерфейс. Значение неинициализированной переменной типа интерфейса равно нулю.

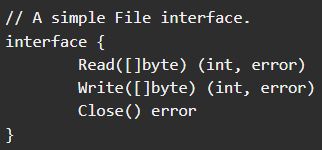


Рисунок 4 – Пример интерфейса в языке *Golang*

В *Golang* реализованы структуры (рисунок 5) [1]. Структура – это последовательность именованных элементов, называемых полями, каждый из которых имеет имя и тип.

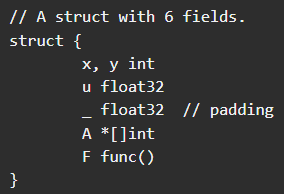


Рисунок 5 – Пример структуры в языке *Golang*

Тип указателя обозначает набор всех указателей на переменные данного типа, называемый базовым типом указателя. Значение неинициализированного указателя равно nil.

Кроме того, в *Golang* можно создавать пользовательские типы данных с помощью ключевого слова *type*. Это позволяет определить новые типы, основанные на существующих или состоящие из нескольких значений (структуры).

Рассмотрим условные операторы языка *Golang*.

1. Оператор *if* (рисунок 6). Оператор *if* используется для выполнения блока кода, если заданное условие истинно.

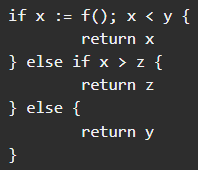


Рисунок 6 – Синтаксис оператора *if*

1. Оператор *switch*. Оператор *switch* позволяет выбрать один из нескольких блоков кода на основе значения выражения. Синтаксис выглядит следующим образом (рисунок 7):

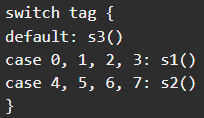


Рисунок 7 – Синтаксис оператора *switch*

1. Операторы сравнения. В *Golang* доступны стандартные операторы сравнения, такие как == (равно), != (не равно), > (больше), < (меньше), >= (больше или равно), <= (меньше или равно).
2. Операторы логического выражения. В *Golang* доступны логические операторы && (логическое И), *||* (логическое ИЛИ), !(логическое НЕ). Они используются для комбинирования и инвертирования логических значений.

В языке программирования Go доступны следующие операторы цикла:

1. Оператор цикла *for*. Оператор *for* в *Go* используется для повторения блока кода определенное количество раз или до выполнения заданного условия. Синтаксис оператора можно увидеть ниже (рисунок 8):

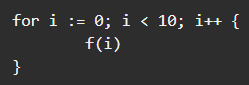


Рисунок 8 – Синтаксис оператора *for*

1. Оператор цикла *while*. В языке *Go* нет оператора *while*, но его функциональность может быть достигнута с использованием оператора *for*. Условие проверяется перед каждой итерацией. Синтаксис можно увидеть ниже (рисунок 9):

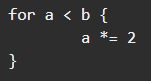


Рисунок 9 – Синтаксис аналога оператора *while*

1. Оператор цикла *do*-*while*. В языке *Go* также нет явного оператора *do*-*while*, но его функциональность может быть достигнута с использованием оператора *for* с бесконечным циклом и выходом из цикла с помощью оператора *break* (рисунок 10).

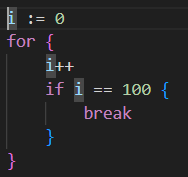


Рисунок 10 – Синтаксис аналога оператора *do­while*

1. Операторы управления циклом. Оператор *break* используется для немедленного выхода из цикла. Оператор *continue* используется для пропуска текущей итерации цикла и перехода к следующей итерации. Пример использования операторов управления циклами *continue* и *break* можно увидеть ниже (рисунок 11):

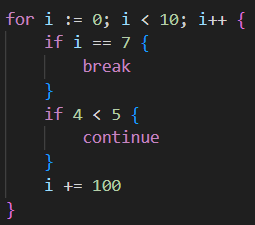


Рисунок 11 – Синтаксис операторов управления циклами

Неотъемлемой частью языка являются пакеты. Пакет *Golang* – это каталог в рабочей области вашего проекта, в котором хранятся один или несколько исходных файлов Golang или других вложенных пакетов *Golang*. В Go каждый исходный файл должен принадлежать пакету. Синтаксис подключения пакетов можно наблюдать на рисунке ниже (рисунок 12):

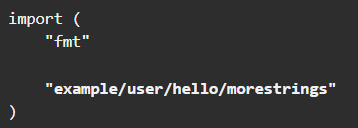


Рисунок 12 – Синтаксис подключения пакетов

В данной главе были рассмотрены все типы, условные операторы, операторы циклов, структуры языка *Golang*, подключения пакетов, которые будут использоваться в ходе анализа данного языка.

1. **ИНСТРУМЕНТАЛЬНАЯ ЯЗЫКОВАЯ СРЕДА**

В качестве языковой среды выбран язык программирования *C++ 20*. Разработка основана на работе с операционной системой *Windows* на *PC.*

Инструментальная среда *C++ 20* представляет собой среду разработки, которая обеспечивает программистам возможности для создания, отладки и тестирования программ на языке программирования C++. Рассмотрим возможности языковой среды.

Инструментальная среда *C++ 20* включает в себя редактор кода (*Visual Studio* *Code*), который предоставляет различные функции, такие как подсветка синтаксиса, автодополнение кода, инструменты отладки, а также инструменты для рефакторинга кода.

Для разработки программ на *C++* *20* необходим компилятор, который может преобразовывать исходный код на *C++* в исполняемый файл (*clang*++, *g*++).

Некоторые инструментальные среды *C++ 20* могут предоставлять встроенные или сторонние утилиты для анализа кода. Эти утилиты могут проверять синтаксис, обнаруживать потенциальные проблемы, предлагать рекомендации по стилю кодирования и выполнению других видов статического анализа.

Инструментальная среда *C++ 20* может иметь интеграцию с системами контроля версий, такими как *Git*. Это позволяет программистам управлять версиями исходного кода, отслеживать изменения и сотрудничать с другими разработчиками.

В данной главе была рассмотрена основная языковая среда, а также была выбрана платформа, на которой будет проводиться анализ.

# ЗАКЛЮЧЕНИЕ

В ходе работы было определено подмножества языка программирования (типы констант, переменных, операторов и функций). В подмножество языка были включены все типы. Были учтены учесть все структуры, представленные в рассматриваемом языке. Учтены подключения пакетов. Операторы цикла *(do...while, for)*. Условные операторы *(if...else, case).* Определена инструментальная языковая среда, т.е. язык программирования и операционная система для разработки. В ходе лабораторной работе дается полное определение подмножества языка программирования, тексты 3-х программ, включающих все элементы этого подмножества. Приводится подробное описание инструментальной языковой среды.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. *Effective Go* [Электронный ресурс]. – Режим доступа: https://go.dev/doc/effective\_go – Дата доступа: 12.02.2024.

# ПРИЛОЖЕНИЕ А (обязательное) Текст программ

Листинг 1 – Example1.go

package example1

import "errors"

func Max(a, b int) int { // вместо int любой числовой тип

if a > b {

return a

}

return b

}

func CountChars(s string) map[rune]int {

cnt := make(map[rune]int, 0)

for \_, rn := range s {

cnt[rn]++

}

return cnt

}

func SliceMax(sl []int) (int, error) {

if len(sl) > 0 {

mx := sl[0]

for i := 0; i < len(sl); i++ {

mx = Max(mx, sl[i])

}

return mx, nil

}

return -1, errors.New("Slice was empty!")

}

func TwoLargestNums(sl []int) ([2]int, error) {

if len(sl) < 2 {

return [2]int{}, errors.New("Slice len < 2")

} else if len(sl) == 2 {

return [2]int(sl), nil

}

ans := [2]int{sl[0], sl[0]}

for \_, num := range sl {

if num >= ans[0] {

ans[1] = ans[0]

ans[0] = num

} else if num > ans[1] && num != ans[0] {

ans[1] = num

}

}

return ans, nil

}

Листинг 2 – Example2.go

package example2

type Shape interface {

Area() float64

}

type Circle struct {

Radius float64

}

type Rectangle struct {

Width float64

Height float64

}

func (c Circle) Area() float64 {

return 3.14 \* c.Radius \* c.Radius

}

func (r Rectangle) Area() float64 {

return r.Width \* r.Height

}

Листинг 3 – Main.go

package main

import (

"examples/example1"

"examples/example2"

"fmt"

"log"

)

func main() {

circle := example2.Circle{Radius: 5}

rectangle := example2.Rectangle{Width: 4, Height: 6}

shapes := []example2.Shape{circle, rectangle}

for \_, shape := range shapes {

switch shape.(type) {

case example2.Circle:

fmt.Println("Circle")

case example2.Rectangle:

fmt.Println("Rectangle")

}

area := shape.Area()

fmt.Printf("Area: %.2f\n", area)

if area > 20 {

break

}

if area < 10 {

continue

}

}

max := example1.Max(10, 20)

fmt.Println("Max:", max)

counts := example1.CountChars("hello")

fmt.Println("Character Counts:", counts)

slice := []int{5, 8, 3, 12, 6}

maxNum, err := example1.SliceMax(slice)

if err != nil {

log.Fatal(err)

}

fmt.Println("Maximum Number in Slice:", maxNum)

twoLargest, err := example1.TwoLargestNums(slice)

if err != nil {

log.Fatal(err)

}

fmt.Println("Two Largest Numbers in Slice:", twoLargest)

}