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 М. А. Щур
Преподаватель Н. Ю. Гриценко

Минск 2025

СОДЕРЖАНИЕ

1 ЦЕЛЬ РАБОТЫ

Цель данной работы заключается в формировании подмножества языка программирования, включая все типы констант, переменных, операторов и функций. В рамках этого подмножества должны быть представлены все доступные типы данных, а также учтены все структуры, доступные в рассматриваемом языке. Также необходимо рассмотреть возможности подключения библиотек (модулей) и других ресурсов.

В работе будут проанализированы операторы циклов, такие как *do...while* и *for*, которые предоставляет язык, а также условные операторы, включая *if...else* и *case*.

Кроме того, будет определена инструментальная языковая среда, включающая язык программирования с указанием версии, операционную систему (например, *Windows* или *Linux*), на которой будет происходить разработка, а также тип компьютера (*PC* или *Macintosh*).

В отчете по лабораторной работе будет представлено полное описание подмножества языка программирования, а также тексты 3 программ, которые включают все элементы этого подмножества. Также будет приведено детальное описание инструментальной языковой среды.

2 ПОДМНОЖЕСТВО ЯЗЫКА ПРОГРАММИРОВАНИЯ

В языке программирования *Fortran* существует множество элементов, которые позволяют разрабатывать различные типы программ. В этом разделе мы подробно рассмотрим подмножество языка, включающее типы данных, переменные, операторы циклов, условные операторы и структуры данных, с конкретными примерами.

2.1 Типы данных в языке программирования *Fortran*

Fortran поддерживает несколько базовых типов данных, которые можно использовать для объявления переменных.

1 Целые числа *(INTEGER)* – числа со знаком, размер которых зависит от платформы (обычно 32 бита). Диапазон от -2147483648 до 2147483647.

2 Вещественные числа *(REAL)* – используется для хранения чисел с плавающей запятой. Размер зависит от платформы (обычно 32 бита). Диапазон от -3.4E+38 до 3.4E+38.

3 Двойная точность *(DOUBLE PRECISION)* – используется для хранения чисел с плавающей запятой двойной точности. Размер зависит от платформы (обычно 64 бита). Диапазон от -1.7E+308 до 1.7E+308.

4 Комплексные числа *(COMPLEX)* – используется для хранения комплексных чисел. Размер зависит от платформы (обычно 64 бита).

5 Строки *(CHARACTER)* – Используется для хранения строковых данных фиксированной длины. Размер зависит от указанной длины.

6 Логические типы *(LOGICAL)* – используется для хранения логических значений (*.TRUE.* или *.FALSE.*) и занимает 32 бита.

2.2 Операторы циклов и условные операторы

Рассмотрим условные операторы языка *Fortran*.

1 Оператор *IF*. Оператор *IF* используется для выполнения блока кода, если заданное условие истинно. Синтаксис представлен на рисунке 1.

```
IF (number > 0) THEN
    PRINT *, "Число положительное"
ELSE IF (number < 0) THEN
    PRINT *, "Число отрицательное"
ELSE
    PRINT *, "Число равно нулю"
END IF
```

Рисунок 1 – Синтаксис оператора *IF*

2 Оператор *SELECT CASE*. Оператор *SELECT CASE* позволяет выбрать один из нескольких блоков кода на основе значения выражения. Синтаксис представлен на рисунке 2.

```
SELECT CASE (score)
    CASE (90:100)
        PRINT *, "Отлично"
    CASE (75:89)
        PRINT *, "Хорошо"
    CASE (50:74)
        PRINT *, "Удовлетворительно"
    CASE DEFAULT
        PRINT *, "Неудовлетворительно"
END SELECT
```

Рисунок 2 – Синтаксис оператора *SELECT CASE*

3 В языке программирования *Fortran* существуют операторы сравнения, такие как == (равно), != (не равно), > (больше), < (меньше), >= (больше или равно), <= (меньше или равно), которые позволяют сравнивать значения различных типов данных. Эти операторы возвращают логическое значение (*.TRUE.* или *.FALSE.*) в зависимости от результата сравнения.

4 Операторы логического выражения. В *Fortran* доступны логические операторы *.AND.* (логическое И), *.OR.* (логическое ИЛИ), *.NOT.* (логическое НЕ). Они используются для комбинирования и инвертирования логических значений.

В языке программирования *Fortran* доступны следующие операторы цикла:

1 Оператор цикла *DO*. Оператор *DO* в *Fortran* используется для повторения блока кода определенное количество раз или до выполнения заданного условия. Синтаксис оператора можно увидеть на рисунке 3.

```
DO i = 1, 10
    sum = sum + i
END DO
```

Рисунок 3 – Синтаксис оператора *DO*

2 Оператор цикла *DO WHILE*. используется для создания цикла, который повторяет выполнение блока кода, пока заданное условие истинно. Синтаксис можно увидеть на рисунке 4.

```
DO WHILE (number >= 0)
        sum = sum + number
        PRINT *, "Текущая сумма:", sum
        PRINT *, "Введите следующее число
        READ *, number
END DO
```

Рисунок 4 – Синтаксис оператора *DO WHILE*

3 Операторы управления циклом. Оператор *EXIT* используется для немедленного выхода из цикла. Оператор *CYCLE* используется для пропуска текущей итерации цикла и перехода к следующей итерации. Пример использования операторов *CYCLE* и *EXIT* можно увидеть на рисунке 5.

```
DO i = 1, 10
    IF (i == 5) THEN
        PRINT *, "Выход из цикла на i =", i
        EXIT
    END IF

    IF (MOD(i, 2) == 0) THEN
        PRINT *, "Пропуск четного числа i =", i
        CYCLE    END IF

    PRINT *, "Нечетное число i =", i
END DO
```

Рисунок 5 – Синтаксис операторов управления циклами

2.3 Структуры

В *Fortran* реализованы пользовательские структуры [1]. Структуры (или пользовательские типы данных) позволяют объединять различные типы данных в единую единицу. Это особенно полезно, когда необходимо хранить связанные данные, которые могут иметь разные типы. Пользовательские структуры позволяют разработчикам создавать более сложные и организованные модели данных. Пример объявления структуры представлен на рисунке 6.

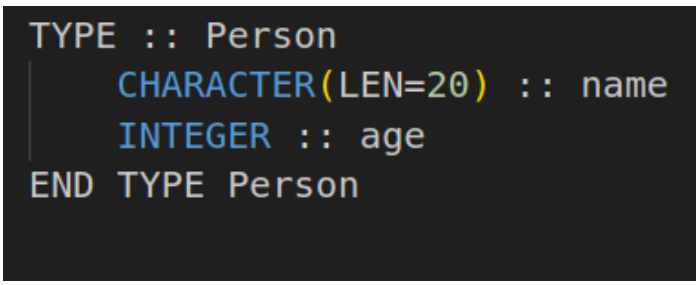

Рисунок 6 – Пример структуры в языке *Fortran*

Массивы в *Fortran* – это структуры данных, которые позволяют хранить наборы значений одного и того же типа в виде единой переменной. Это удобно для работы с большими объемами данных и упрощает код, так как позволяет обращаться к элементам массива по индексам. Индексация массивов в *Fortran* начинается с 1, то есть первый элемент массива имеет индекс 1. Пример использования массивов представлен на рисунке 7.

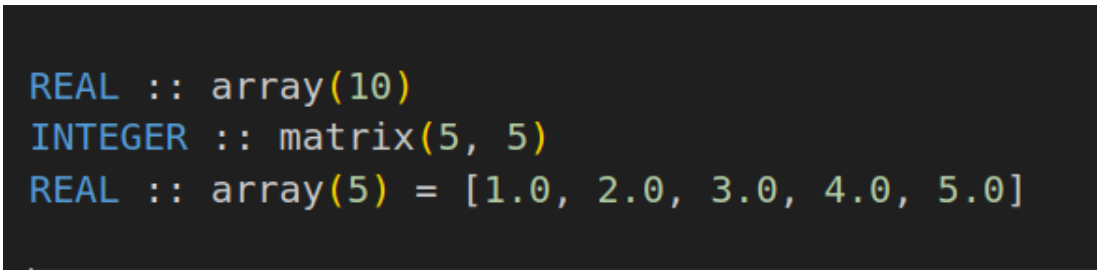

Рисунок 7 – Пример массива в языке *Fortran*

В Fortran существуют функции, которые позволяют организовывать код, делая его более структурированным и переиспользуемым. Функции в *Fortran* могут принимать аргументы, выполнять вычисления и возвращать результаты. Функции могут принимать как входные, так и выходные аргументы. Для входных аргументов используется спецификатор *INTENT*(*IN*), для выходных – *INTENT*(*OUT*).

3 ИНСТРУМЕНТАЛЬНАЯ ЯЗЫКОВАЯ СРЕДА

В качестве языковой среды выбран язык программирования *Go* (версия 1.23.5). Разработка основана на работе с операционной системой *Ubuntu* на ПК.

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

Инструментальная среда *Go* включает в себя редактор кода, такой как *Visual Studio Code* или *GoLand*, которые предоставляют различные функции, включая подсветку синтаксиса, автодополнение кода, инструменты отладки и рефакторинга.

Для разработки программ на *Go* необходим компилятор, который может преобразовывать исходный код на *Go* в исполняемый файл. Встроенный компилятор *Go* позволяет создавать исполняемые файлы для различных платформ и архитектур, что упрощает кросс-компиляцию.

Инструментальная среда *Go* также включает в себя средства для тестирования, которые позволяют разработчикам писать и запускать тесты для проверки функциональности кода. Это особенно важно для обеспечения качества и надежности программного обеспечения.

Кроме того, *Go* имеет встроенную поддержку управления зависимостями, которая упрощает процесс работы с внешними библиотеками и пакетами.

Инструментальная среда *Go* 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 для языка *Go*, а также была выбрана платформа, на которой будет проводиться разработка и анализ.

ЗАКЛЮЧЕНИЕ

В ходе работы было определено подмножество языка программирования *Fortran*, включая типы, константы, переменные, операторы и функции. В подмножество языка были включены все основные типы данных, такие как *INTEGER*, *REAL*, *CHARACTER* и *LOGICAL*, операторы цикла и условные операторы. Учтены структурные типы, включая пользовательские структуры, что позволяет группировать связанные данные. Определена инструментальная языковая среда, т.е. язык программирования и операционная система для разработки. В ходе лабораторной работы дается полное определение подмножества языка программирования, тексты трёх программ, включающих все элементы этого подмножества. Приводится подробное описание инструментальной языковой среды.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Fortran High-performance parallel programming language [Электронный ресурс]. – Режим доступа: https://fortran-lang.org/. – Дата доступа: 23.01.2025.

ПРИЛОЖЕНИЕ А
(обязательное)
Текст программ

Листинг 1 – module1.f90

```
module module1
  implicit none

  type person
    character(len=50) :: name
    integer :: age
    real :: height
  end type person

contains

  function create_person(name_in, age_in, height_in) result(person_out)
    implicit none
    character(len=*), intent(in) :: name_in
    integer, intent(in) :: age_in
    real, intent(in) :: height_in
    type(person) :: person_out
    person_out%name = name_in
    person_out%age = age_in
    person_out%height = height_in
  end function create_person

  subroutine print_person_info(person_in)
    implicit none
    type(person), intent(in) :: person_in
    print *, "Имя:", person_in%name
    print *, "Возраст:", person_in%age
    print *, "Рост:", person_in%height
  end subroutine print_person_info

  subroutine do_loop_example(n)
    implicit none
    integer, intent(in) :: n
    integer :: i
    do i = 1, n
      print *, "Итерация цикла DO:", i
    end do
  end subroutine do_loop_example

  subroutine do_while_loop_example(n)
    implicit none
    integer, intent(in) :: n
    integer :: i = 1
    do while (i <= n)
      print *, "Итерация цикла DO WHILE:", i
      i = i + 1
    end do
  end subroutine do_while_loop_example
end module module1
```

Листинг 2 – module2.f90

```
module module2
  implicit none

  integer, dimension(1:10) :: numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

```
contains

  function sum_array(arr) result(sum)
    implicit none
    integer, dimension(:), intent(in) :: arr
    integer :: sum
    integer :: i
    sum = 0
    do i = 1, size(arr)
      sum = sum + arr(i)
    end do
  end function sum_array

  subroutine select_case_demo(value)
    implicit none
    integer, intent(in) :: value
    select case (value)
      case (1:5)
        print *, "Значение находится в диапазоне 1-5"
      case (6:10)
        print *, "Значение находится в диапазоне 6-10"
      case default
        print *, "Значение не находится в указанных диапазонах"
    end select
  end subroutine select_case_demo

  subroutine loop_control_example(n)
    implicit none
    integer, intent(in) :: n
    integer :: i
    do i = 1, n
      if (i == 5) then
        cycle
      end if
      if (i == 8) then
        exit
      end if
      print *, "Итерация с EXIT и CYCLE:", i
    end do
  end subroutine loop_control_example

end module module2
```

Листинг 3 – main.f90

```
program main
  use module1
  use module2
  implicit none
  type(person) :: my_person
  integer :: array_sum
  my_person = create_person("Иван Иванов", 30, 1.85)
  call print_person_info(my_person)
  array_sum = sum_array(numbers)
  print *, "Сумма элементов массива:", array_sum
  call select_case_demo(7)
  call do_loop_example(5)
  call do_while_loop_example(3)
  call loop_control_example(10)

end program main
```