Министерство науки и высшего образования Российской Федерации

Новосибирский Государственный технический университет

Кафедра автоматизированных систем управления



**Отчет по лабораторной работе №2**

**Вариант №7**

**по дисциплине «Архитектура средств вычислительной техники»**

**«Многосвязные списковые структуры»**

**Вариант–2.2**

Выполнили

студенты группы АВТ-813:

Кинчаров Данил

Пайхаев Алексей

Чернаков Кирилл

Преподаватель:

Ландовский Владимир Владимирович,

к.т.н., доцент кафедры АСУ

г. Новосибирск

2020 год

# Цель работы

Познакомиться с многосвязными структурами данных, получить представление об использовании таких структур для описания реальных данных, приобрести навыки программирования функций доступа к данным.

# Схемы структур данных

Таблица 1 – вариант задания.

|  |  |  |  |
| --- | --- | --- | --- |
| Вариант | Задача | Организация списка | Функции |
| 2.2 | Состав изделий | Непосредственно в файлах | Упаковка файлов, вывод полного состава |

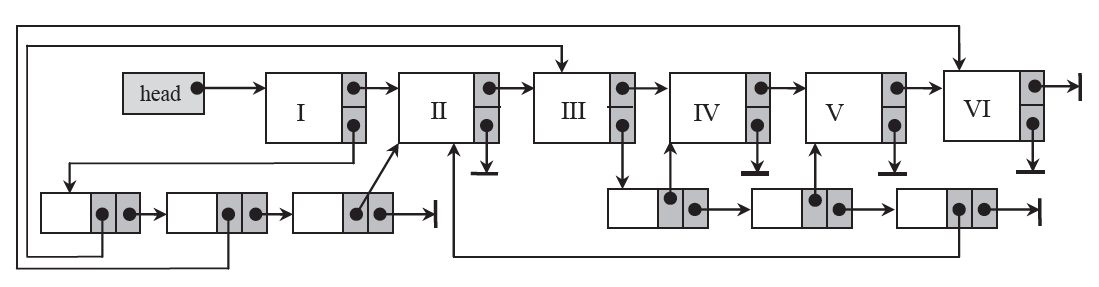


Рис. 1 – Многосвязная списковая структура

# Описание формата использующихся файлов



Рис. 2 – Структура одной записи в файле деталей

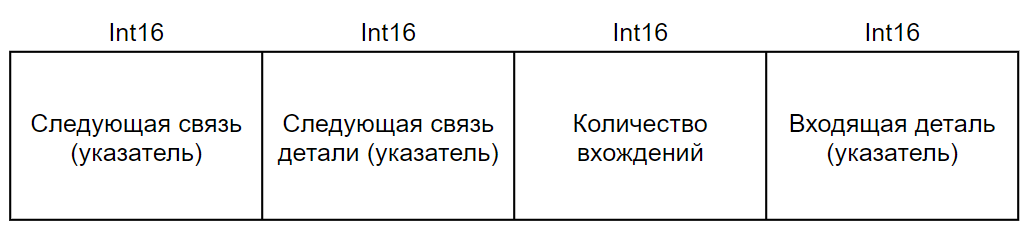


Рис. 3 – Структура одной записи в файле связей

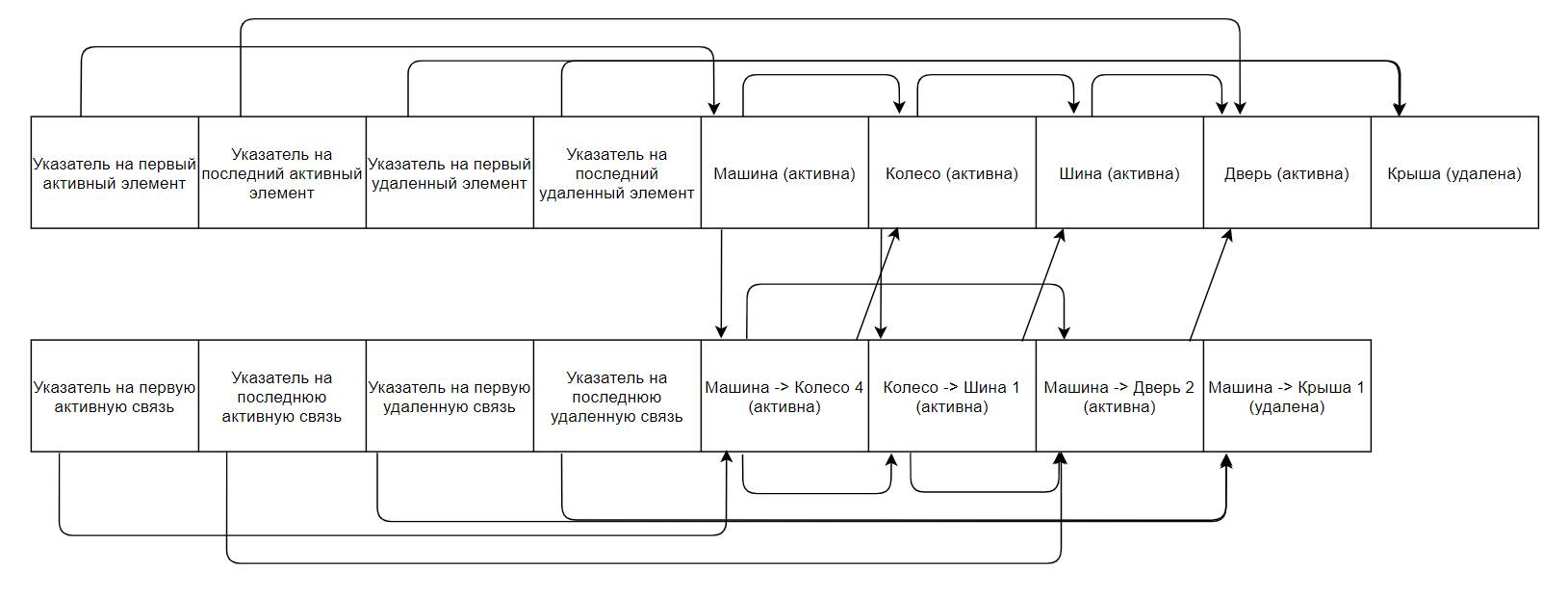


Рис. 4 – Структура файлов

В файле деталей записаны 5 деталей, в файле связей – 4 связи. Машина, Колесо, Шина и Дверь являются активными и составляют список активных деталей. Деталь Крыша удалена и составляет список удаленных деталей. Машина включает в себя детали Колесо и Дверь. Деталь Колесо включает в себя деталь Шина. Деталь крыша исключена и удалена из Машины.

В файле связей содержится 4 связи. Две из них относятся к Машине, одна к Колесу и последняя связь удалена. Связи, относящиеся к Машине, включают 3 указателя, в этом случае первый указатель указывает на следующую активную связь в списке связей, второй указывает на следующую активную связь Машины, последний указывает на деталь, с которой связана Машина (Колесо и Дверь).

# Тексты программ

List.js

const ConnectionManager = require( './ConnectionManager' )

const NodeManager = require( './NodeManager' )

const { ipcMain } = require('electron')

class List {

    constructor( nodeManager, connectionManager ) {

        this.nodeManager = nodeManager

        this.connectionManager = connectionManager

    }

    async add( name, nameNodeTo = '', number = 1 ) {

        let node = {

            name: name,

            connection: -1,

            next: -1

        }

        await this.addListElement( this.nodeManager, node, this.nodeManager.activeInformation )

        if ( nameNodeTo !== '' ) {

            await this.include( name, nameNodeTo, number )

        }

    }

    async findNode( name, pointer = -1 ) {

      if ( pointer === -1) {

        await this.nodeManager.readInformation()

        pointer = this.nodeManager.activeInformation.head

      }

      let node = await this.nodeManager.get( pointer )

      if ( node.name !== name ) {

          return await this.findNode( name, node.next )

      } else {

          return pointer

      }

    }

    async addListElement( manager, element, information ) {

        let pointer = -1

        if( information.head === -1 ) {

            information.head = manager.informationSize

            information.tail = manager.informationSize

            await manager.writeInformation()

            pointer = await manager.put( element )

        } else {

            pointer = await manager.put( element )

            element = await manager.get( information.tail )

            element.next = pointer

            await manager.put( element, information.tail)

            information.tail = pointer

            await manager.writeInformation()

        }

        return pointer

    }

    async include( name, nameNodeTo, number ) {

        let connection = {

                node: await this.findNode( name ),

                amount: number,

                nextNodeConnection: -1,

                next: -1,

        }

        let pointer = await this.addListElement( this.connectionManager, connection, this.connectionManager.activeInformation )

        await this.attachConnection( pointer, nameNodeTo )

    }

    async attachConnection( connectionPointer, name ) {

        let nodePointer = await this.findNode( name )

        let node = await this.nodeManager.get( nodePointer )

        if ( node.connection === -1 ) {

            node.connection = connectionPointer

            await this.nodeManager.put( node, nodePointer )

        } else {

            let pointer = node.connection

            let connection  = await this.connectionManager.get( node.connection )

            while ( connection.nextNodeConnection !== -1 )

            {

                pointer = connection.nextNodeConnection

                connection = await this.connectionManager.get( connection.nextNodeConnection )

            }

            connection.nextNodeConnection = connectionPointer

            await this.connectionManager.put( connection, pointer )

        }

    }

    async exclude( name, nameFrom, number = 0 ) {

        let pointer = await this.findNode( name )

        let pointerFrom = await this.findNode( nameFrom )

        let node = await this.nodeManager.get( pointerFrom )

        if ( node.connection === -1 ) {

            return

        }

        let connection = await this.connectionManager.get( node.connection )

        let currentConnectionPointer = node.connection

        let prevConnectionPointer = node.connection

        while ( connection.node !== pointer ) {

            prevConnectionPointer = currentConnectionPointer

            currentConnectionPointer = connection.nextNodeConnection

            if ( connection.nextNodeConnection === -1) {

                return

            }

            connection = await this.connectionManager.get( connection.nextNodeConnection )

        }

        if ( number >= connection.amount || !number ) {

            if ( node.connection === currentConnectionPointer ) {

                node.connection = connection.nextNodeConnection

                await this.nodeManager.put( node, pointerFrom )

            } else {

                let nextConnectionPointer = connection.nextNodeConnection

                connection = await this.connectionManager.get( prevConnectionPointer )

                connection.nextNodeConnection = nextConnectionPointer

                await this.connectionManager.put( connection, prevConnectionPointer )

            }

            await this.delete( currentConnectionPointer, this.connectionManager )

        } else {

            connection.amount -= number

            await this.connectionManager.put( connection, currentConnectionPointer )

        }

    }

    async excludeFromAll( pointer ) {

        await this.nodeManager.readInformation()

        let node = await this.nodeManager.get( pointer )

        let fromPointer = this.nodeManager.activeInformation.head

        let nodeFrom = await this.nodeManager.get( fromPointer )

        await this.exclude( node.name, nodeFrom.name )

        while ( nodeFrom.next !== -1 ) {

            nodeFrom = await this.nodeManager.get( nodeFrom.next )

            await this.exclude( node.name, nodeFrom.name )

        }

    }

    async deleteNode( name, pointer = -1 ) {

        await this.nodeManager.readInformation()

        if ( pointer === -1 ) {

            pointer = await this.findNode( name )

            await this.excludeFromAll( pointer )

            if ( pointer === this.nodeManager.activeInformation.head ) {

                await this.cleanList()

                return

            }

        }

        let node = await this.nodeManager.get( pointer )

        let connectionPointer = node.connection

        while( connectionPointer !== -1 ) {

            let connection = await this.connectionManager.get( connectionPointer )

            let prevConnectionPointer = connectionPointer

            if( !( await this.isNodeNeeded( connection.node ) ) ) {

                await this.deleteNode( '', connection.node )

            }

            connectionPointer = connection.nextNodeConnection

            await this.delete( prevConnectionPointer , this.connectionManager )

        }

        await this.delete( pointer, this.nodeManager )

    }

    async isNodeNeeded( pointer ) {

        await this.connectionManager.readInformation()

        let connection = await this.connectionManager.get( this.connectionManager.activeInformation.head )

        while ( connection.next !== -1 ) {

            if ( connection.node === pointer ) {

                return true

            } else {

                connection = await this.connectionManager.get( connection.next )

            }

        }

        return false

    }

    async delete( pointer, manager ) {

        await manager.readInformation()

        let currentPointer = manager.activeInformation.head

        let prevPointer = currentPointer

        let element = await manager.get( currentPointer )

        while ( currentPointer !== pointer ) {

            prevPointer = currentPointer

            currentPointer = element.next

            element = await manager.get( currentPointer )

        }

        if ( currentPointer === manager.activeInformation.tail ) {

            manager.activeInformation.tail = prevPointer

            await manager.writeInformation()

        }

        let nextPointer = element.next

        await this.addListElement( manager, element, manager.deletedInformation )

        element = await manager.get( prevPointer )

        element.next = nextPointer

        await manager.put( element, prevPointer )

    }

    async cleanList() {

        await this.nodeManager.resetFile()

        await this.connectionManager.resetFile()

    }

    async getNodeString( name ) {

        let nodePointer = await this.findNode( name )

        let node = await this.nodeManager.get( nodePointer )

        let connectionPointer = node.connection

        let result = `Node : ${ node.name } \nContains : `

        if ( connectionPointer === -1 ) {

            result += `empty, `

        }

        while ( connectionPointer !== -1 ) {

            let connection = await this.connectionManager.get( connectionPointer )

            node = await this.nodeManager.get( connection.node )

            result += `${ node.name } ${ connection.amount }, `

            connectionPointer = connection.nextNodeConnection

        }

        return result.replace( /, $/g, '\n' )

    }

    async getListString() {

        await this.nodeManager.readInformation()

        if ( this.nodeManager.activeInformation.head === -1 ) {

            return 'empty\n'

        }

        let node = await this.nodeManager.get( this.nodeManager.activeInformation.head )

        let result = ``

        result += await this.getNodeString( node.name )

        while ( node.next !== -1 ) {

            node = await this.nodeManager.get( node.next )

            result += await this.getNodeString( node.name )

        }

        return result

    }

}

module.exports = List

Manager.js

const BinaryFile = require("binary-file")

class Manager {

  constructor( fileName ) {

    this.fileName = fileName

    this.file = {}

    this.fileOpen = false

    this.informationSize = 8

    this.setDefault()

  }

  async openFile( ) {

    if ( !this.fileOpen ) {

      this.file = new BinaryFile( this.fileName, 'w+' )

      await this.file.open()

      this.fileOpen = true

    }

  }

  async closeFile() {

    if ( this.fileOpen ) {

      await this.file.close()

      this.fileOpen = false

    }

  }

  setDefault() {

    this.activeInformation = {

      head: -1,

      tail: -1

    }

    this.deletedInformation = {

      head: -1,

      tail: -1

    }

  }

  async resetFile() {

    this.setDefault()

    await this.closeFile()

    await this.openFile()

    await this.writeInformation()

  }

  async writeInformation() {

    await this.openFile()

    await this.setCursorPosition( 0 )

    await this.file.writeInt16( this.activeInformation.head )

    await this.file.writeInt16( this.deletedInformation.head )

    await this.file.writeInt16( this.activeInformation.tail )

    await this.file.writeInt16( this.deletedInformation.tail )

  }

  async readInformation() {

    await this.openFile()

    await this.setCursorPosition( 0 )

    this.activeInformation.head = await this.file.readInt16() || -1

    this.deletedInformation.head = await this.file.readInt16() || -1

    this.activeInformation.tail = await this.file.readInt16() || -1

    this.deletedInformation.tail = await this.file.readInt16() || -1

  }

  async setCursorPosition( position = -1 ) {

    await this.openFile()

    let size = await this.file.size()

    position = position === -1 ? size : position

    this.file.seek( position )

    return position

  }

  async put( element, position = -1 ) {

    position = await this.setCursorPosition( position )

    await this.write( element )

    return position

  }

  async get( position = -1) {

    await this.setCursorPosition( position )

    return await this.read()

  }

}

module.exports = Manager

NodeManager.js

const Manager = require( './Manager' )

class NodeManager extends Manager {

  constructor( dataSize, fileName ) {

    super( fileName )

    this.dataSize = dataSize

  }

  async read() {

    await this.openFile()

    let node = {

        name: ( await this.file.readString( this.dataSize ) ).replace(/[\u0000-\u0009]/g, ''),

        connection: await this.file.readInt16(),

        next: await this.file.readInt16()

    }

    return node

  }

  async write( node ) {

    await this.openFile()

    let buffer = Buffer.alloc( this.dataSize )

    buffer.write( node.name )

    await this.file.write( buffer )

    await this.file.writeInt16( node.connection )

    await this.file.writeInt16( node.next )

  }

}

module.exports = NodeManager

ConnectionManager.js

const Manager = require( './Manager' )

class ConnectionManager extends Manager {

  constructor( fileName ) {

    super( fileName )

  }

  async read() {

    await this.openFile()

    let connection = {

      node: await this.file.readInt16(),

      amount: await this.file.readInt16(),

      nextNodeConnection: await this.file.readInt16(),

      next: await this.file.readInt16()

    }

    return connection

  }

  async write( connection ) {

    await this.openFile()

    await this.file.writeInt16( connection.node )

    await this.file.writeInt16( connection.amount )

    await this.file.writeInt16( connection.nextNodeConnection )

    await this.file.writeInt16( connection.next )

  }

}

module.exports = ConnectionManager

# Пример работы программы с описанием интерфейса пользователя

Для наглядности после каждого взаимодействия с программой выводим полный список деталей.

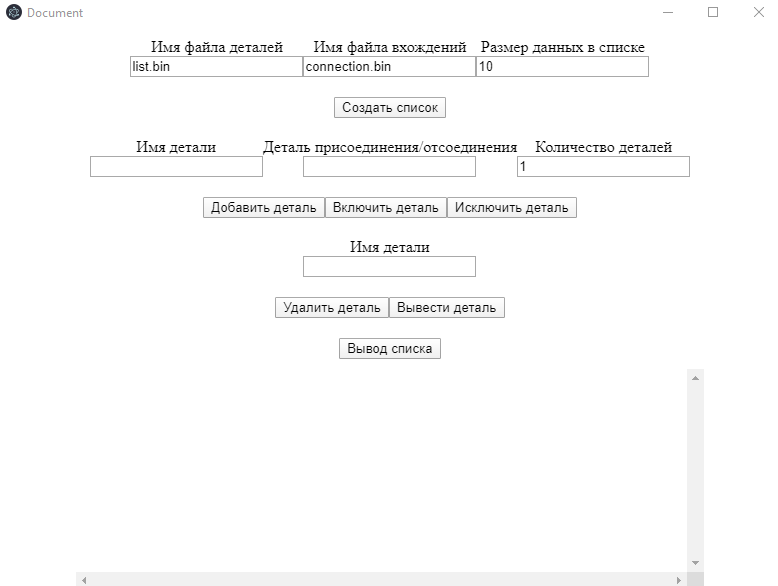


Рис. 5 – Графический пользовательский интерфейс при запуске программы



Рис. 6 – Создание списка (список пустой)



Рис. 7 – Добавление первого элемента (Car)

Рис. 8 – Добавление второго элемента, связанного с первым (Wheel)



Рис. 9 – Добавление третьего элемента, связанного с первым (Door)



Рис. 10 – Исключение элемента Door из Car в количестве 2 штук



Рис. 11 – Включение уже имеющейся детали Door в состав Car (создание багажника)

Рис. 12 – Удаление элемента Wheel



Рис. 13 – Вывод состава Car