

Task 1.

Изучить механизм интеропа между языками, попробовать у себя вызывать C/C++ код из Java и C#. В отчёте описать логику работы, сложности и ограничения этих механизмов.

Процесс интеропа – это возможность вызвать код, написанный на одной языке, из другого языка.

C++ - C#

Пишем код на C++ для h-файла и его реализацию. В результате получаем динамически подключаемую библиотеку .dll.

library.cpp

```
#include "library.h"

#include <iostream>

int Sum(int n, int m) {
    return n+m;
}
```

library.h

```
#ifndef PROJECT1_LIBRARY_H
#define PROJECT1_LIBRARY_H

extern "C" {
    int __declspec(dllexport) Sum(int, int);
}

#endif //PROJECT1_LIBRARY_H
```

Получается библиотека libproject1.dll, загружаем ее по пути к файлу.

```
using System;
using System.Runtime.InteropServices;

namespace implement1
{
    internal static class Program
    {
        [DllImport("C:\\Users\\Я\\Desktop\\TechProg\\lab-1\\task1\\project1\\cmake-build-debug\\libproject1.dll",
            CallingConvention = CallingConvention.Cdecl)]
        private static extern int Sum(int n, int m);

        public static void Main(string[] args)
        {
            Console.WriteLine(Sum(2, 5));
        }
    }
}
```

library.cpp

```
#include "library.h"

JNIEXPORT jint JNICALL Java_my_sum_Main_Add
    (JNIEnv * env, jobject obj, jint n, jint m){
    int sum = (int)(n+m);
    return (jint)sum;
}
```

library.h

```
#include "C:\TP\jni.h"

#ifndef _Included_my_sum_Main
#define _Included_my_sum_Main
#ifdef __cplusplus
extern "C" {
#endif
    JNIEXPORT jint JNICALL Java_my_sum_Main_Add
        (JNIEnv *, jobject, jint, jint);

#ifdef __cplusplus
};
#endif
#endif
```

Получается библиотека libTask1LibC2.dll, загружаем ее по пути к файлу.

```
package my.sum;

public class Program {
    static
    {
        System.load( filename: "C:\\Teta\\TP\\lab-1\\Task1LibC2\\cmake-build-debug\\libTask1LibC2.dll");
    }

    native public static int Add(int n, int m);

    public static void main(String[] args) { System.out.println(Add( n: 3, m: 4)); }
}
```

У данного способа есть как плюсы (возможность не переписывать код с одного языка на другой), так и минусы (замедляется выполнение программы, код сложнее писать и тестировать, возрастает вероятность ошибок).

Task 2.

Написать немного кода на Scala и F# с использованием уникальных возможностей языка - Pipe operator, Discriminated Union, Computation expressions. Вызвать написанный код из обычных соответствующих ООП языков (Java и C#) и посмотреть во что превращается написанный ранее код после декомпиляции в них.

Scala – Java

Pipe operator, Discriminated Union на Scala:

```
object Task2 {  
  def plus6(i: Int): Int = i + 6  
  def multiply3(i: Int): Int = i * 3  
  def Func(f: Int): Int = f.pipe(plus6).pipe(multiply3)  
  
  sealed trait Shapes  
  case class Circle(radius: Int) extends Shapes  
  case class Rectangle(length1: Int, length2: Int) extends Shapes  
  case class Rhomb(length: Int) extends Shapes  
  
  def Perimeter(shapes: Shapes) : Double = shapes match {  
    case Circle(radius) => 2*3.14*radius  
    case Rectangle(length1, length2) => 2*length1*length2  
    case Rhomb(length) => 4*length  
  }  
}
```

Часть кода Java после декомпиляции:

```
public final class Task2$ {  
  public static final Task2$ MODULE$ = new Task2$();  
  
  public int plus6(final int i) { return i + 6; }  
  
  public int multiply3(final int i) { return i * 3; }  
  
  public int Func(final int f) {  
    return BoxesRunTime.unboxToInt(.MODULE$.pipe$extension(scala.util.package.chaining..MODULE$.scalaUtilCh  
      return MODULE$.plus6(i);  
    })), (i) -> {  
      return MODULE$.multiply3(i);  
    });  
  }  
}
```

```

public double Perimeter(final Task2.Shapes shapes) {
    double var2;
    if (shapes instanceof Task2.Circle) {
        Task2.Circle var5 = (Task2.Circle)shapes;
        int radius = var5.radius();
        var2 = 6.28D * (double)radius;
    } else if (shapes instanceof Task2.Rectangle) {
        Task2.Rectangle var7 = (Task2.Rectangle)shapes;
        int length1 = var7.length1();
        int length2 = var7.length2();
        var2 = (double)(2 * length1 * length2);
    } else {
        if (!(shapes instanceof Task2.Rhomb)) {
            throw new MatchError(shapes);
        }

        Task2.Rhomb var10 = (Task2.Rhomb)shapes;
        int length = var10.length();
        var2 = (double)(4 * length);
    }

    return var2;
}

```

F# - C#

Pipe operator, Discriminated Union, Competition expression на F#:

Pipe operator:

```

module pipeOp =
    int -> int
    let plus6 n=n+6
    int -> int
    let multiply3 n=n*3
    int -> int
    let func(x: int) =
        x:int
        |> plus6:int
        |> multiply3

```

Competition expression:

```

module compExp =
    (int * int * int * int * int * int * int * int * int * int) list
    let list = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
    seq<int>
    let doubles =
        seq {
            for i in 1..10
            -> i*7
        }
    for x in doubles do
        printfn $"%i{x}"

```

Discriminated Union:

```
type Shapes =  
    | Circle of double  
    | Rectangle of double * double  
    | Rhomb of double  
  
module DiscUn =  
    Shapes -> float  
  
    let Perimeter perimeterShapes =  
        match perimeterShapes with  
        | Circle radius -> 2.0*3.14*radius  
        | Rectangle (length1, length2) -> 2.0 * (length1 + length2)  
        | Rhomb length -> 4.0 * length
```

Часть кода C# после декомпиляции:

```
namespace project2_2  
{  
    [Serializable]  
    [StructLayout(LayoutKind.Auto, CharSet = CharSet.Auto)]  
    [DebuggerDisplay("{__DebugDisplay(),nq}")]  
    [CompilationMapping(SourceConstructFlags.SumType)]  
    public abstract class Shapes : IEquatable<Shapes>, IStructuralEquatable, IComparable  
    {  
        public static class Tags  
        {  
            public const int Circle = 0;  
  
            public const int Rectangle = 1;  
  
            public const int Rhomb = 2;  
        }  
  
        [Serializable]  
        [SpecialName]  
        [DebuggerTypeProxy(typeof(Circle@DebugTypeProxy))]  
        [DebuggerDisplay("{__DebugDisplay(),nq}")]  
        public class Circle : Shapes  
        {  
            [DebuggerBrowsable(DebuggerBrowsableState.Never)]  
            [CompilerGenerated]  
            [DebuggerNonUserCode]  
            internal readonly double item;  
  
            [CompilationMapping(SourceConstructFlags.Field, 0, 0)]  
            [CompilerGenerated]  
            [DebuggerNonUserCode]  
            public double Item  
            {  
                [CompilerGenerated]  
                [DebuggerNonUserCode]
```

Pipe operator:

```
[CompilationMapping(SourceConstructFlags.Module)]
public static class pipeOp
{
    public static int plus6(int n)
    {
        return n + 6;
    }

    public static int multiply3(int n)
    {
        return n * 3;
    }

    public static int func(int x)
    {
        return (x + 6) * 3;
    }
}
```

Competition expression:

```
[CompilationMapping(SourceConstructFlags.Module)]
public static class compExp
{
    [Serializable]
    [SpecialName]
    [StructLayout(LayoutKind.Auto, CharSet = CharSet.Auto)]
    [CompilationMapping(SourceConstructFlags.Closure)]
    internal sealed class doubles@7 : GeneratedSequenceBase<int>
    {
        [DebuggerBrowsable(DebuggerBrowsableState.Never)]
        [CompilerGenerated]
        [DebuggerNonUserCode]
        public IEnumerator<int> @enum;

        [DebuggerBrowsable(DebuggerBrowsableState.Never)]
        [CompilerGenerated]
        [DebuggerNonUserCode]
        public int pc;

        [DebuggerBrowsable(DebuggerBrowsableState.Never)]
        [CompilerGenerated]
        [DebuggerNonUserCode]
        public int current;

        public doubles@7(IEnumerator<int> @enum, int pc, int current)
        {
            this.@enum = @enum;
            this.pc = pc;
            this.current = current;
            base..ctor();
        }
    }
}
```

Discriminated Union:

```
[CompilationMapping(SourceConstructFlags.Module)]
public static class DiscUn
{
    public static double Perimeter(Shapes perimeterShapes)
    {
        if (!(perimeterShapes is Shapes.Rectangle))
        {
            if (!(perimeterShapes is Shapes.Rhomb))
            {
                Shapes.Circle circle = (Shapes.Circle)perimeterShapes;
                return 2.0 * 3.14 * circle.item;
            }
            Shapes.Rhomb rhomb = (Shapes.Rhomb)perimeterShapes;
            return 4.0 * rhomb.item;
        }
        Shapes.Rectangle rectangle = (Shapes.Rectangle)perimeterShapes;
        double item = rectangle.item2;
        double item2 = rectangle.item1;
        return 2.0 * (item2 + item);
    }
}
```

Вывод: так как Scala и F# имеют уникальные возможности, которых нет в Java и C#, код после декомпиляции на этих языках становится большим по объему и непривычен для восприятия.

Task 3.

Написать алгоритм обхода графа (DFS и BFS) на языке Java, собрать в пакет и опубликовать. Использовать в другом проекте на Java/Scala этот пакет. Повторить это с C#/F#. В отчёте написать про алгоритм работы пакетных менеджеров, особенности их работы в C# и Java мирах.

Java:

Алгоритмы DFS и BFS:

```
package com.company;

import java.util.Iterator;
import java.util.LinkedList;

public class Search {
    public int U;
    public LinkedList<Integer> adj[];

    public Search(int u){
        U=u;
        adj=new LinkedList[u];
        for (int i=0; i<u; ++i){
            adj[i] = new LinkedList();
        }
    }

    public void addEdge(int u, int w) { adj[u].add(w);}
```

```

public void BFS(int a){
    boolean wasVisited[] = new boolean[U];
    LinkedList<Integer> queue = new LinkedList<Integer>();
    wasVisited[a]=true;
    queue.add(a);

    while(queue.size() != 0){
        a = queue.poll();
        System.out.print(a+" ");

        Iterator<Integer> i = adj[a].listIterator();
        while (i.hasNext()){
            int k = i.next();
            if (!wasVisited[k]){
                wasVisited[k] = true;
                queue.add(k);
            }
        }
    }
}
}

```

```

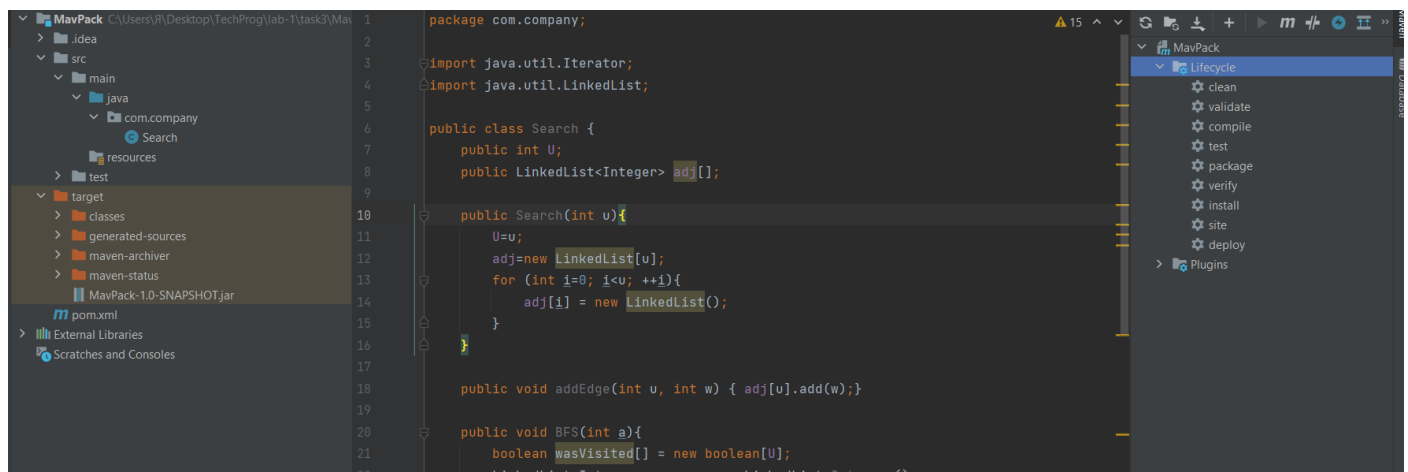
public void DFSUtil(int u, boolean wasVisited[]){
    wasVisited[u] = true;
    System.out.print(u + " ");

    Iterator<Integer> i = adj[u].listIterator();
    while (i.hasNext()){
        int k = i.next();
        if (!wasVisited[k]){
            DFSUtil(k, wasVisited);
        }
    }
}

public void DFS(int u){
    boolean wasVisited[] = new boolean[U];
    DFSUtil(u, wasVisited);
}
}

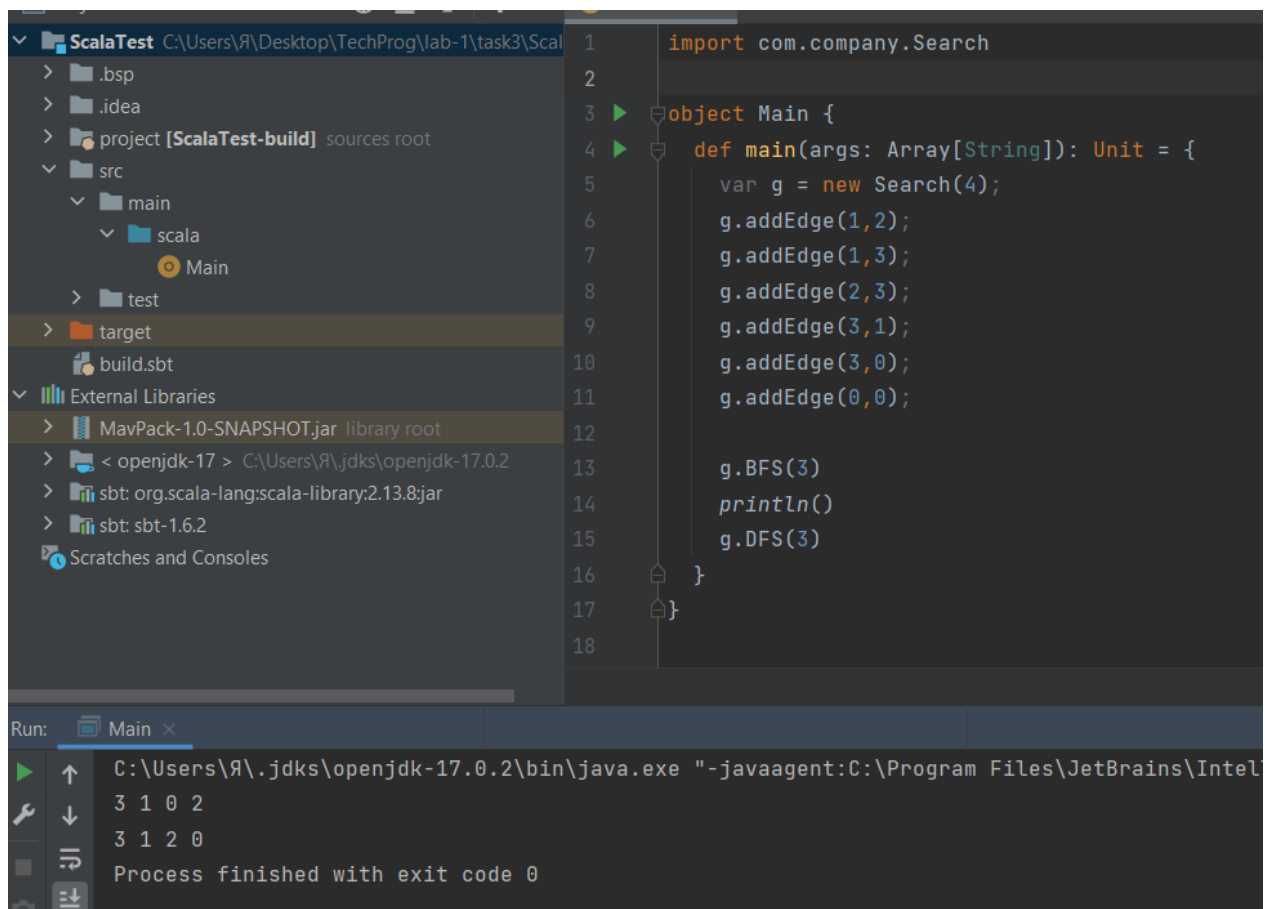
```

Создадим пакет в Maven (Maven -> раскрываем наш пакет -> Lifecycle -> package).



Получили jar файл, который можно использовать в других проектах на Scala и Java.

Scala:



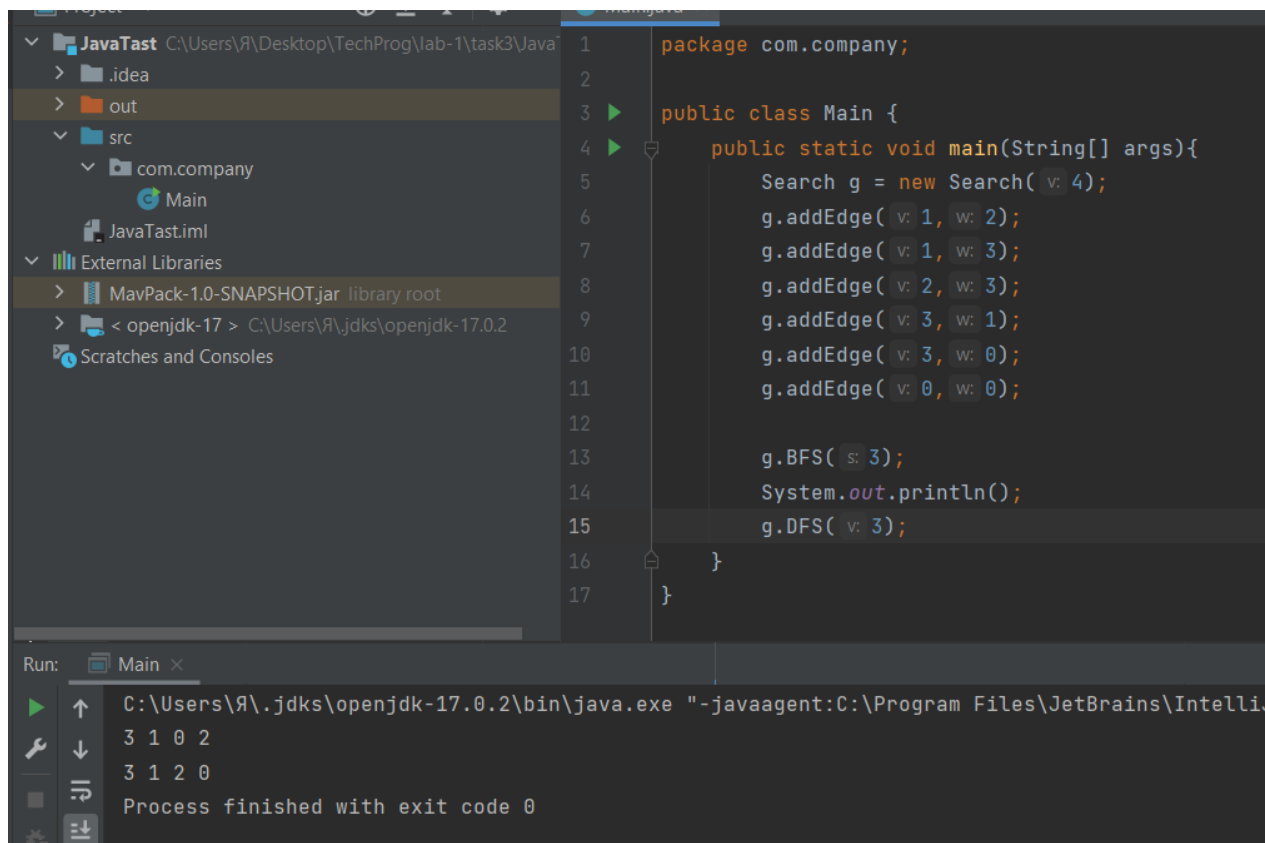
The screenshot shows the IntelliJ IDEA IDE with a Scala project named 'ScalaTest'. The left sidebar displays the project structure, including the 'src/main/scala' directory and the 'Main' object. The main editor window shows the following Scala code:

```
1 import com.company.Search
2
3 object Main {
4   def main(args: Array[String]): Unit = {
5     var g = new Search(4);
6     g.addEdge(1,2);
7     g.addEdge(1,3);
8     g.addEdge(2,3);
9     g.addEdge(3,1);
10    g.addEdge(3,0);
11    g.addEdge(0,0);
12
13    g.BFS(3)
14    println()
15    g.DFS(3)
16  }
17 }
```

The bottom panel shows the Run configuration for 'Main' and the execution output:

```
Run: Main
C:\Users\Я\jdk\openjdk-17.0.2\bin\java.exe "-javaagent:C:\Program Files\JetBrains\IntelliJ
3 1 0 2
3 1 2 0
Process finished with exit code 0
```

Java:



The screenshot shows the IntelliJ IDEA IDE with a Java project named 'JavaTest'. The left sidebar displays the project structure, including the 'src/main/java' directory and the 'Main' class. The main editor window shows the following Java code:

```
1 package com.company;
2
3 public class Main {
4   public static void main(String[] args){
5     Search g = new Search(4);
6     g.addEdge(1, 2);
7     g.addEdge(1, 3);
8     g.addEdge(2, 3);
9     g.addEdge(3, 1);
10    g.addEdge(3, 0);
11    g.addEdge(0, 0);
12
13    g.BFS(3);
14    System.out.println();
15    g.DFS(3);
16  }
17 }
```

The bottom panel shows the Run configuration for 'Main' and the execution output:

```
Run: Main
C:\Users\Я\jdk\openjdk-17.0.2\bin\java.exe "-javaagent:C:\Program Files\JetBrains\IntelliJ
3 1 0 2
3 1 2 0
Process finished with exit code 0
```

C#/F#:

Алгоритмы DFS и BFS на C#:

```
namespace CodeC
{
    public class Search
    {
        private static void Main(){
        }

        private int _u;
        private List<int>[] adj;

        public Search(int u)
        {
            _u = u;
            adj = new List<int>[u];
            for (int i = 0; i < u; i++)
            {
                adj[i] = new List<int>();
            }
        }

        public void AddEdge(int u, int w)
        {
            adj[u].Add(w);
        }
    }
}
```

```
public void BFS(int a)
{
    bool[] wasVisited = new bool[_u];
    for (int i = 0; i < _u; i++)
    {
        wasVisited[i] = false;
    }

    LinkedList<int> queue = new LinkedList<int>();
    wasVisited[a] = true;
    queue.AddLast(a);
}
```

```

while (queue.Any())
{
    a = queue.First();
    Console.Write(a + " ");
    queue.RemoveFirst();
    var list = adj[a];

    foreach (var value:int in list.Where(val:int => !wasVisited[val]))
    {
        wasVisited[value] = true;
        queue.AddLast(value);
    }
}
}

```

2 usages

```

public void DFSUtil(int u, bool[] wasVisited)
{
    wasVisited[u] = true;
    Console.Write(u + " ");
    List<int> uList = adj[u];
    foreach (var k:int in uList)
    {
        if (!wasVisited[k])
        {
            DFSUtil(k, wasVisited);
        }
    }
}

public void DFS(int u)
{
    bool[] wasVisited = new bool[_u];
    DFSUtil(u, wasVisited);
}
}

```

Собираем код в пакет NuGet с помощью Rider (Advanced Build Action -> Pack Select Project) и публикуем на сайте NuGet, далее NuGet пакет можно найти в Rider и подключить его.

The screenshot shows the Visual Studio IDE with an F# project named 'TestF'. The project structure on the left includes 'Dependencies' with '.NET 6.0', 'Assemblies', 'Packages' (CodeC/1.0.0, FSharp.Core/6.0.2), 'Frameworks', and 'Imports'. The main editor displays the 'Program.fs' file with the following code:

```
1 open CodeC
2 let g = new Search(4)
3 g.AddEdge(1,2);
4 g.AddEdge(1,3);
5 g.AddEdge(2,3);
6 g.AddEdge(3,1);
7 g.AddEdge(3,0);
8 g.AddEdge(0,0);
9
10 g.BFS(3)
11 printfn ""
12 g.DFS(3)
```

The bottom of the image shows the 'Run' window with the command 'TestF' and the output path 'C:/Users/Я/Desktop/TechProg/lab-1/task3/TestF/TestF/bin/Debug/net6.0/TestF.exe'. The output shows two lines of numbers: '3 1 0 2' and '3 1 2 0', followed by the message 'Process finished with exit code 0.'

Task 4.

Изучить инструменты для оценки производительности в C# и Java. Написать несколько алгоритмов сортировок (и взять стандартную) и запустить бенчмарки. В отчёт написать про инструменты для бенчмаркинга, их особенности, анализ результатов проверок.

C#:

Sorts.cs:

```
using System;
using BenchmarkDotNet.Attributes;

namespace projectC {
    [MemoryDiagnoser]
    public class Sorts
    {
        [Params(100, 10000)]
        public int N;
        public int[] array;

        [GlobalSetup]
        public void Setup()
        {
            array = new int[N];
            var random = new Random();
            for (var i = 0; i < array.Length; i++)
            {
                array[i] = random.Next();
            }
        }

        [Benchmark]
        public void InsertionSort()
        {
            Insertion.sort(array);
        }
    }
}
```

```

[Benchmark]
public void InsertionSort()
{
    Insertion.sort(array);
}

[Benchmark]
public void BubbleSort()
{
    Bubble.sort(array);
}

[Benchmark]
public void MergeSort()
{
    Merge.sort(array, 0, array.Length - 1);
}

[Benchmark]
public void StandartSort()
{
    Array.Sort(array);
}
}
}

```

Running.cs:

```

using BenchmarkDotNet.Running;

namespace projectC
{
    public class Running
    {
        private static void Main()
        {
            BenchmarkRunner.Run<Sorts>();
        }
    }
}

```

Когда запускаем класс, BenchmarkDotNet сначала выполняет подготовку (определяет количество итераций и оценивает накладную работу), а затем переходит к измерению.

Результаты:

Method	N	Mean	Error	StdDev	Gen 0	Allocated
InsertionSort	100	122.8 ns	2.50 ns	2.87 ns	-	-
BubbleSort	100	5,260.6 ns	100.75 ns	123.73 ns	-	-
MergeSort	100	3,124.9 ns	62.24 ns	155.00 ns	3.8223	8,000 B
StandartSort	100	343.3 ns	6.60 ns	6.17 ns	-	-
InsertionSort	10000	11,248.9 ns	207.83 ns	213.42 ns	-	-
BubbleSort	10000	46,916,917.6 ns	931,708.12 ns	1,985,545.97 ns	-	48 B
MergeSort	10000	469,823.2 ns	8,986.31 ns	8,405.80 ns	512.2070	1,072,016 B
StandartSort	10000	61,663.0 ns	1,043.26 ns	975.87 ns	-	-

Java:

Проводим измерения с помощью Java Microbenchmark Harness (JMH).

Program.java:

```
package Bench;
import org.openjdk.jmh.annotations.*;

import java.util.Arrays;
import java.util.Random;
import java.util.concurrent.TimeUnit;
import Sorts.*;

@State(Scope.Thread)
public class Program {
    @Param({"100", "10000"})
    private int N;
    private int[] array;

    @Setup(Level.Invocation)
    public void setUp() {
        array = new int[N];
        Random rand = new Random();
        for (int i = 0; i < array.length; i++) {
            array[i] = rand.nextInt();
        }
    }
}
```

```

@Benchmark
@BenchmarkMode(Mode.AverageTime)
@OutputTimeUnit(TimeUnit.NANOSECONDS)
@Fork(value = 1)
@Measurement(iterations = 10)
@Warmup(iterations = 1)
public void Insertion(){
    Insection.sort(array);
}

@Benchmark
@BenchmarkMode(Mode.AverageTime)
@OutputTimeUnit(TimeUnit.NANOSECONDS)
@Fork(value = 1)
@Measurement(iterations = 10)
@Warmup(iterations = 1)
public void Bubble() { Bubble.sort(array); }

```

```

@Benchmark
@BenchmarkMode(Mode.AverageTime)
@OutputTimeUnit(TimeUnit.NANOSECONDS)
@Fork(value = 1)
@Measurement(iterations = 10)
@Warmup(iterations = 1)
public void Merge() { Merge.sort(array, l: 0, r: array.length - 1); }

@Benchmark
@BenchmarkMode(Mode.AverageTime)
@OutputTimeUnit(TimeUnit.NANOSECONDS)
@Fork(value = 1)
@Measurement(iterations = 10)
@Warmup(iterations = 1)
public void Stand(){
    Arrays.sort(array);
}
}

```

Runner.java:

```

package Bench;

public class Runner {
    public static void main(String[] args) throws Exception {
        org.openjdk.jmh.Main.main(args);
    }
}

```

Результаты:

Benchmark	(N)	Mode	Cnt	Score	Error	Units
Bench.Program.Bubble	100	avgt	10	6626,836 ±	474,541	ns/op
Bench.Program.Bubble	10000	avgt	10	23202100,365 ±	1924355,426	ns/op
Bench.Program.Insertion	100	avgt	10	2182,941 ±	389,997	ns/op
Bench.Program.Insertion	10000	avgt	10	8512746,811 ±	1018820,835	ns/op
Bench.Program.Merge	100	avgt	10	6853,596 ±	985,502	ns/op
Bench.Program.Merge	10000	avgt	10	1293362,768 ±	226077,414	ns/op
Bench.Program.Stand	100	avgt	10	2677,727 ±	153,111	ns/op
Bench.Program.Stand	10000	avgt	10	503022,478 ±	20184,327	ns/op

Результаты работы бенчмарков помогают оценить скорость работы сортировок и использование ими памяти, т. е. их эффективность.

Task 5.

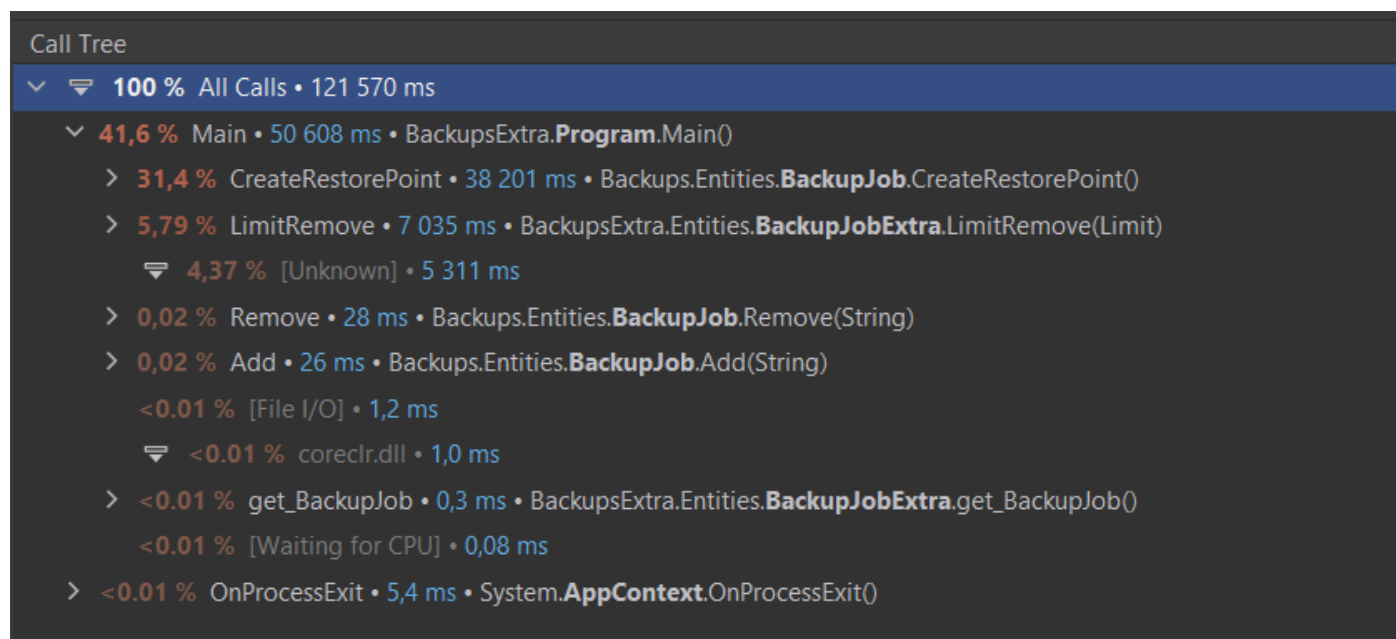
Используя инструменты dotTrace, проанализировать работу написанного кода для бэкапов. Необходимо написать сценарий, когда в цикле будет выполняться много запусков, будут создаваться и удаляться точки. Проверить два сценария: с реальной работой с файловой системой и без неё. В отчёте необходимо проанализировать полученные результаты, сделать вывод о написанном коде.

В цикле 10000 раз выполняется создание точек, добавление в них файлов, удаление точек и удаление файлов из джобы.

Измерения dotTrace без файловой системы:

Call Tree	
100 %	All Calls • 4 400 ms
49,6 %	Main • 2 181 ms • BackupsExtra. Program .Main()
49,4 %	CreateRestorePointInVirtualMemory • 2 172 ms • Backups.Entities. BackupJob .CreateRestorePointInVirtualMemory()
49,4 %	BackupInVirtualMemory..ctor • 2 171 ms • Backups.Entities. BackupInVirtualMemory ..ctor(List)
<0,01 %	[File I/O] • 0,009 ms
0,11 %	Add • 4,8 ms • Backups.Entities. BackupJob .Add(String)
0,05 %	Remove • 2,0 ms • Backups.Entities. BackupJob .Remove(String)
0,03 %	RemoveFromVirtualMemory • 1,5 ms • BackupsExtra.Entities. BackupJobExtra .RemoveFromVirtualMemory()
0,03 %	[Unknown] • 1,4 ms
0,19 %	OnProcessExit • 8,3 ms • System. AppContext .OnProcessExit()

Измерения dotTrace с файловой системой:



Выводы:

- основные затраты при работе с файловой системой идут на архивацию файлов, немало затрат уходит на удаление точек
- при работе без файловой системы на основные затраты идут на чтение и запись информации о точках при их создании