Fourth homework

A Byte Of Assembler

Task: Various numbers, variant 291 (11, 21)

Autor: Davydov Vyacheslav Olegovich

BPI208 group

Task

> Various numbers * 1 Complex (real and imaginary parts - a pair of real numbers) * 2 Simple fractions (numerator, denominator - a pair of integers) * 3 Polar coordinates (angle [radian] - real; coordinates of the end point on the plane

---

Functions common to all alternatives

Converting each value to a real number equivalent to the written one. For example, for a complex number it is carried out according to the formula: sqrt (d^2 + i^2)), and for polar coordinates - distance.

Basic alternatives

> *Procedural*
>
> > - presence, absence of abstract data types - boolean
>
> *Object-oriented*
>
> > - inheritance: single, multiple, interface - enumerated
>
> *Functional*
>
> > - typing - enumerated type = strong, dynamic
> > - lazy evaluation support - boolean

Functional

After placing the data in the container, it is necessary to process it in accordance with the task variant. The processed data are then entered into a separate result file.

Remove from the container those elements for which the value obtained using the function common to all alternatives is less than the arithmetic mean for all elements of the container, obtained using the same function. Move other elements to the beginning container while maintaining order.

Testing

The initial data for testing is contained in the `output` directory.

Tests explanation

> ***Number index is a digit between 1 and 3 where:***
>
> - *1 - it is a complex number that has two parameters: > * x - real part > * y - imaginary part > * #### real value of this number count according to formula: sqrt(x^2+y^2)*
> - *2 - it is a fraction. It has a parameters named: > * a - a numerator > * b - a denominator > * #### real value can be retrived by division parametr a on b*
> - *3 - it is a polar coordinate. It has: > * r - radius > * phi - angle > * #### It's real value is just parametr r*

Required metrics that determine the characteristics of the program, for various test runs.

> The program contains 0 interface modules (header files) and 7 implementation modules (files with the definition of software objects): * delete.asm - 2.46 KB * inrnd.asm - 6.50 KB * input.asm - 7.56 KB * output.asm - 9.49 KB * main.asm - 7.60 KB * real.asm - 4.23 KB * macros.mac - 4.71 KB

Metric	Value

Metric	Value
The total size of the source code of the program	42.5 KB
Release build executable size (GCC, Linux)*	32.8 KB

Compilation

```
$ make
```

Then:

```
$ ./task -f input.txt output01.txt output2.txt
```

- for file input or:

```
$ ./task -n 1000 output01.txt output2.txt
```

- for random generated input

Statistics of different approaches

Procedural (C ++)

Program runtime on different sizes of input data:

Memory: 16.2 KB

Number of numbers	Running time, seconds
`10`	`0.04`
`100`	`0.11`
`1000`	`0.38`
`5000`	`1.35`

Object Oriented (C++)

Program runtime on different sizes of input data:

Memory: 16.7 KB

Number of numbers	Running time, seconds
`10`	`0.004`
`100`	`0.006`
`1000`	`0.04`
`5000`	`0.26`

Dynamic typing (Python)

Program runtime on different sizes of input data:

Memory: 7.29 KB

Number of numbers	Running time, seconds
`10`	`0.006`
`100`	`0.02`
`1000`	`0.2`
`5000`	`1.27`

Assembly

Program runtime on different sizes of input data:

Memory: 42.5 KB

Number of numbers	Running time, seconds
`10`	`0.007`
`100`	`0.0116`
`1000`	`0.019`
`5000`	`0.13`

Difference between procedural, object-oriented, dynamic typing with an assembly program

> Of course, a program in Assembler is more difficult to write than a program in a high-level language. However, Assembler also has advantages.
>
> > First, a program written in a high-level language is still translated into an assembler program, and in a very suboptimal way. That is, an assembler program will almost always be run faster and take up significantly less memory.
> >
> > Second, access to many hardware resources can only be obtained using Assembler. Please note that an **assembler** program can be written in any editor!
>
> As a result we get that: * The program, written in NASM, is significantly faster than the previous three programs. * The original texts, compared to previous homework, take up much more memory, which may be due to the fact that the code on NASM turns out to be very voluminous

Additional functionality

Additional tasks were implemented: modular structure, explanatory comments and handling incorrect numbers in the input data

***** Details:

```
 gcc --version:
gcc (Ubuntu 9.3.0-17ubuntu1~20.04) 9.3.0
Copyright (C) 2019 Free Software Foundation, Inc.
```

```
 nasm --version:
NASM version 2.14.02
```

```
 lsb_release -a:
No LSB modules are available.
Distributor ID: Ubuntu
Description:    Ubuntu 20.04 LTS
Release:        20.04
Codename:       focal
```

```
 uname -a:
Linux riizeron 5.10.16.3-microsoft-standard-WSL2 #1 SMP Fri Apr 2 22:23:49 UTC 2021 x86_64 x86_64 x86_64 GNU/Linux
```

CPU: Intel(R) Pentium(R) Silver N5000 CPU @ 1.10GHz