

Benchmark Performa YAZA vs Bahasa Lain

Ringkasan Eksekutif

Pengukuran ulang mencakup 10 bahasa: YAZA, C#/.NET, Python, JavaScript/Node.js, Ruby, Java, PHP, Go, Nim, dan Kotlin/JVM. Ketujuh workload lulus validasi output kecil (77 dari 77). R tidak diukur karena Rscript belum tersedia. C/Clang tetap ditampilkan sebagai acuan native terpisah.

YAZA tidak menjadi yang tercepat pada satu pun dari tujuh workload setelah Go dan Nim ditambahkan. Go mengalahkan YAZA pada 7/7 workload; Nim mengalahkannya pada 6/7. Go memimpin pada cetak baris, penjumlahan, dan Fibonacci; Nim memimpin pada Hello World, hitung prima, konversi angka ke teks, dan pemindaian list. YAZA lebih cepat daripada C#/.NET, Python, JavaScript, Ruby, Java, PHP, dan Kotlin pada semua workload yang diukur.

Waktu median YAZA untuk Fibonacci hanya sekitar 1,08 kali waktu Go, dan pada penjumlahan sekitar 1,11 kali. Selisih terbesar terhadap bahasa tercepat ada pada konversi angka ke teks: Nim sekitar 3,42 kali lebih cepat. Implementasi C/Clang manual juga lebih cepat dari YAZA sekitar 1,33 sampai 2,41 kali; C bukan peserta ranking.

Keunggulan YAZA atas beberapa runtime lain mungkin terkait executable native dan biaya startup yang lebih kecil. Karena pengukuran mencakup seluruh umur proses, hasil Java, Kotlin/JVM, dan .NET juga mencakup startup VM/runtime. Data ini belum memisahkan biaya startup dari waktu kerja. Kesimpulannya: YAZA menunjukkan potensi performa awal, tetapi hasil ini tidak mendukung klaim bahwa YAZA lebih cepat daripada bahasa native Go/Nim secara umum.

Generated: 2026-10-28 12:57:48 WIB

Ruang Lingkup dan Metode

This is a local command-line microbenchmark on one Apple Silicon Mac. Each sample starts a fresh process and measures wall-clock time from process launch through exit with Python perf_counter_ns. Runtime startup, program work, and shutdown are included. Source compilation/build is a separate one-time measurement. Output for timing runs is redirected to /dev/null; the print workload still performs every formatting and write call. Each implementation first passed the same small-output checks.

Medians use 7 measured samples for Hello World and 7 for each other workload. Hello World used 2 separate-process warm-ups; other workloads used 1. Warm-ups may populate OS caches but do not preserve a running VM or JIT between samples. Java/.NET/Kotlin process startup and per-process JIT costs remain part of these results. They are useful for CLI response comparisons; they do not represent long-lived server throughput.

Workloads: Hello World process start; print 50,000 formatted lines to a discard sink; sum integers 1..10,000,000; naive recursive Fibonacci(34); count primes through 100,000 by trial division; convert 1..250,000 to decimal text and count characters; create a 64-element dynamic list and scan it 30,000 times.

The main versus tables compare YAZA with all installed language implementations represented in this run. Hand-written C compiled with Clang is separated below as a technical native baseline and is excluded from language rankings, because YAZA also compiles to native code through C and Clang.

Median waktu (ms)

Workload	n	YAZA	.NET C#	Python	JavaScript	Ruby	Java	PHP	Go	Nim	Kotlin
Hello process start	0	3.277	23.201	13.556	39.259	28.864	27.057	42.280	2.321	1.734	43.530
Print lines (discarded)	50000	5.564	53.871	21.917	79.185	35.710	78.941	61.900	4.121	5.521	81.404
Integer sum loop	10000000	6.454	33.712	358.907	92.773	147.852	38.089	71.232	5.789	6.447	42.786
Recursive Fibonacci	34	17.440	39.430	769.832	76.124	315.006	40.252	217.411	16.204	27.420	46.217
Prime count (trial division)	100000	5.704	28.235	108.911	48.491	84.493	34.152	58.679	4.445	4.385	39.641
Number to text + length	250000	16.313	30.804	42.286	45.251	37.375	47.288	45.484	5.042	4.768	51.057
Dynamic list create + scan	30000	4.306	30.831	73.175	47.864	72.441	36.974	50.658	2.949	2.526	47.355

Perbandingan Relatif

Ratio = runtime median ÷ YAZA median. Below 1.00 means that runtime completed this workload faster than YAZA; above 1.00 means slower. C/Clang is deliberately excluded from this ranking and appears in its own reference table. These are ratios of this machine's measurements, not general language rankings.

Workload	YAZA	.NET C#	Python	JavaScript	Ruby	Java	PHP	Go	Nim	Kotlin
Hello process start	1.00×	7.08×	4.14×	11.98×	8.81×	8.26×	12.90×	0.71×	0.53×	13.28×
Print lines (discarded)	1.00×	9.68×	3.94×	14.23×	6.42×	14.19×	11.13×	0.74×	0.99×	14.63×
Integer sum loop	1.00×	5.22×	55.61×	14.38×	22.91×	5.90×	11.04×	0.90×	1.00×	6.63×
Recursive Fibonacci	1.00×	2.26×	44.14×	4.36×	18.06×	2.31×	12.47×	0.93×	1.57×	2.65×
Prime count (trial division)	1.00×	4.95×	19.09×	8.50×	14.81×	5.99×	10.29×	0.78×	0.77×	6.95×
Number to text + length	1.00×	1.89×	2.59×	2.77×	2.29×	2.90×	2.79×	0.31×	0.29×	3.13×
Dynamic list create + scan	1.00×	7.16×	16.99×	11.11×	16.82×	8.59×	11.76×	0.68×	0.59×	11.00×

Referensi Native C/Clang (di luar ranking bahasa)

C is shown only as a low-level native reference. It is not counted as YAZA's language competitor. YAZA ÷ C above 1.00 means the hand-written C sample ran faster in this workload.

Workload	YAZA (ms)	C/Clang (ms)	YAZA ÷ C
Hello process start	3.277	2.030	1.61×
Print lines (discarded)	5.564	4.150	1.34×
Integer sum loop	6.454	2.679	2.41×
Recursive Fibonacci	17.440	11.741	1.49×
Prime count (trial division)	5.704	4.278	1.33×
Number to text + length	16.313	8.796	1.85×
Dynamic list create + scan	4.306	2.425	1.78×

Waktu Perintah Build (satu kali)

These are wall times for the build commands in this run. Existing compiler and package caches, SDK state, and outputs can affect the numbers, so they are not cold-build estimates. Interpreted source runtimes need no explicit compile step in this suite.

Toolchain	Build time
YAZA	446.8 ms
.NET	679.7 ms
Java	309.9 ms
Go	149.5 ms
Nim	474.2 ms
Kotlin	2042.9 ms
C/Clang native reference (excluded)	45.9 ms

Kesimpulan

Pada tujuh workload CLI yang diukur, ranking membandingkan YAZA dengan: .NET C#, Python, JavaScript, Ruby, Java, PHP, Go, Nim, Kotlin. YAZA tercepat pada 0 dari 7 workload di antara bahasa-bahasa tersebut. Go lebih cepat daripada YAZA pada tujuh dari tujuh workload, sedangkan Nim lebih cepat pada enam dari tujuh; pada Fibonacci, Go memimpin dan YAZA lebih cepat daripada Nim. Rasio runner-up terhadap YAZA berkisar 0.29× hingga 0.93×; di bawah 1.00× berarti runner-up lebih cepat. Pemenang per workload: Hello process start: Nim (1.734 ms); Print lines (discarded): Go (4.121 ms); Integer sum loop: Go (5.789 ms); Recursive Fibonacci: Go (16.204 ms); Prime count (trial division): Nim (4.385 ms); Number to text + length: Nim (4.768 ms); Dynamic list create + scan: Nim (2.526 ms). Hand-written C yang dikompilasi Clang tetap lebih cepat sekitar 1.33× sampai 2.41×; hasil itu disajikan sebagai baseline native terpisah, bukan peserta ranking. Ini menunjukkan masih ada jarak terhadap implementasi C manual pada workload kecil ini. Hasil ini berlaku untuk program CLI proses-baru dengan startup sampai proses selesai, bukan ukuran throughput server atau latency request HTTP. R hanya dihitung bila Rscript tersedia. Karena itu, kesimpulan ini terbatas pada workload, versi runtime, dan mesin yang tercatat di laporan.

Interpretasi dan prospek YAZA

YAZA jauh lebih cepat daripada .NET/C#, Python, JavaScript, Ruby, Java, PHP, dan Kotlin pada mayoritas atau seluruh workload di suite ini. Hal itu masuk akal untuk sebagian hasil karena YAZA menghasilkan executable native melalui C/Clang, sedangkan beberapa pembanding menjalankan VM atau interpreter; waktu startup proses juga dihitung. Namun Go dan Nim yang juga menghasilkan executable native mengungguli YAZA pada hampir semua workload. Jadi pengukuran ini tidak mendukung klaim bahwa YAZA secara umum lebih cepat daripada bahasa native lain. Hasil ini tetap menunjukkan potensi awal: YAZA mendekati Go pada penjumlahan dan Fibonacci, mengalahkan Nim pada Fibonacci, dan mengungguli runtime terkelola/interpretasi di workload yang diukur. Untuk disebut menjanjikan sebagai bahasa masa depan, diperlukan bukti lebih luas tentang performa lintas aplikasi, memori, keamanan tipe dan runtime, diagnostik, kestabilan compiler, portabilitas, pustaka standar, tooling, serta penggunaan proyek nyata. Kesimpulan saat ini: ada sinyal performa yang menarik, tetapi belum cukup untuk memproyeksikan kesiapan atau keunggulan umum.

Versi Runtime

Runtime/compiler	Version reported by the local command
YAZA compiler (project dist)	YAZA native compiler 0.1.0 (self-hosted YAZA source)
dotnet	8.0.127
Python	Python 3.9.6
Node.js	v26.0.0
Ruby	ruby 2.6.10p210 (2022-04-12 revision 67958) [universal.arm64e-darwin25]
Java	openjdk version "17.0.14" 2025-01-21; OpenJDK Runtime Environment JBR-17.0.14+1-1367.22-nomod (build 17.0.14+1-b1367.22)
PHP	PHP 8.5.6 (cli) (built: May 5 2026 21:19:36) (NTS); Copyright (c) The PHP Group
R	not installed / not found in PATH
Go	go version go1.27.3 darwin/arm64
Nim	Nim Compiler Version 2.2.12 [MacOSX: arm64]; Compiled at 2026-09-08
Kotlin compiler	info: kotlinc-jvm 2.4.21 (JRE 27.0.1.0.3)

Rust was not implemented in this comparison. R source is included in the code appendix, but Rscript was unavailable and R was not timed.

Lingkungan dan Batasan

- Machine: macOS-26.5-arm64-arm-64bit (arm64).
- Runtime versions are recorded in results/environment.json; raw samples and medians are in results/results.json and results/results.csv.
- R and Rust are counted only if their runtimes are available. The R implementation is included under sources/r/bench.R; Rust has no implementation in this suite.

- YAZA emits C and compiles to a native executable through the installed compiler. The hand-written C/Clang version is therefore a native technical reference, shown separately and excluded from the language ranking.
- Different standard output buffering, integer/string implementations, garbage collectors, and JIT behavior are part of the language/runtime result. The loop, recurrence, prime test, list values, and expected outputs are aligned, but this is not a language-neutral compiler shootout.
- Go, Nim, and Kotlin toolchains were installed for this external benchmark. Kotlin ran as a JVM application; the Java runtime version used to launch it and the Kotlin compiler version are listed above. No YAZA project files were changed.
- Short CLI tasks are dominated by process startup. CPU results are specific to this machine, OS build, power/thermal state, and installed runtime configuration; rerun `python3 run_benchmarks.py` to reproduce in the current environment.

Lampiran: Kode Sumber Workload

Semua listing berikut adalah implementasi yang divalidasi untuk workload yang sama. Kotlin menggunakan target JVM; C/Clang adalah baseline native terpisah dan tidak masuk ranking bahasa. Jika R tidak tercantum di tabel, implementasi R berikut hanya disertakan sebagai referensi karena `Rscript` tidak tersedia saat pengukuran.

YAZA

File: `sources/yaza/bench.yz`

```
fn fib(n: Number) -> Number:
  if n <= 1:
    return n
  return fib(n - 1) + fib(n - 2)

fn count_primes(limit: Number) -> Number:
  var count: Number = 0
  for candidate in 2..(limit + 1):
    var divisor: Number = 2
    var prime: Boolean = true
    while divisor * divisor <= candidate:
      if candidate % divisor == 0:
        set prime = false
        break
      set divisor = divisor + 1
    if prime:
      set count = count + 1
  return count

start:
  let task: Text = arg(1)
  if task == "hello":
    say "Hello, world!"
  elif task == "print":
    let n: Number = number_parse(arg(2))
    for i in 1..(n + 1):
      say "line {i}"
  elif task == "sum":
    let n: Number = number_parse(arg(2))
    var total: Number = 0
    for i in 1..(n + 1):
      set total = total + i
    say number_text(total)
  elif task == "fib":
    let n: Number = number_parse(arg(2))
    say number_text(fib(n))
  elif task == "primes":
    let n: Number = number_parse(arg(2))
    say number_text(count_primes(n))
  elif task == "format":
    let n: Number = number_parse(arg(2))
```

```

var total: Number = 0
for i in 1..(n + 1):
    set total = total + text_length(number_text(i))
say number_text(total)
elif task == "list":
    let n: Number = number_parse(arg(2))
    var values: List[Number] = List[Number]()
    for i in 0..64:
        list_push(values, (i * 17) % 127)
    var total: Number = 0
    for pass_index in 0..n:
        for value in values:
            set total = total + value
    say number_text(total)
else:
    say "unknown task"

```

C# (.NET)

File: sources/csharp/Bench.cs

```

using System;
using System.Collections.Generic;

static class Bench
{
    static long Fib(long n) => n <= 1 ? n : Fib(n - 1) + Fib(n - 2);
    static long CountPrimes(long limit)
    {
        long count = 0;
        for (long candidate = 2; candidate <= limit; candidate++)
        {
            long divisor = 2;
            bool prime = true;
            while (divisor * divisor <= candidate)
            {
                if (candidate % divisor == 0) { prime = false; break; }
                divisor++;
            }
            if (prime) count++;
        }
        return count;
    }
    static void Main(string[] args)
    {
        string task = args[0];
        long n = args.Length > 1 ? long.Parse(args[1]) : 0;
        switch (task)
        {
            case "hello": Console.WriteLine("Hello, world!"); break;
            case "print": for (long i = 1; i <= n; i++) Console.WriteLine($"line {i}"); break;
            case "sum":
                long total = 0;
                for (long i = 1; i <= n; i++) total += i;
                Console.WriteLine(total);
                break;
            case "fib": Console.WriteLine(Fib(n)); break;
            case "primes": Console.WriteLine(CountPrimes(n)); break;
            case "format":
                long chars = 0;
                for (long i = 1; i <= n; i++) chars += i.ToString().Length;
                Console.WriteLine(chars);
                break;
            case "list":
                var values = new List<long>(64);

```

```
        for (long i = 0; i < 64; i++) values.Add((i * 17) % 127);
        long listTotal = 0;
        for (long pass = 0; pass < n; pass++) foreach (long value in values) listTotal += value;
        Console.WriteLine(listTotal);
        break;
    default: Console.WriteLine("unknown task"); break;
    }
}
```

Python

File: sources/python/bench.py

```
import sys

def fib(n):
    if n <= 1:
        return n
    return fib(n - 1) + fib(n - 2)

def count_primes(limit):
    count = 0
    for candidate in range(2, limit + 1):
        divisor = 2
        prime = True
        while divisor * divisor <= candidate:
            if candidate % divisor == 0:
                prime = False
                break
            divisor += 1
        if prime:
            count += 1
    return count

task = sys.argv[1]
n = int(sys.argv[2]) if len(sys.argv) > 2 else 0
if task == "hello":
    print("Hello, world!")
elif task == "print":
    for i in range(1, n + 1):
        print(f"line {i}")
elif task == "sum":
    total = 0
    for i in range(1, n + 1):
        total += i
    print(total)
elif task == "fib":
    print(fib(n))
elif task == "primes":
    print(count_primes(n))
elif task == "format":
    total = 0
    for i in range(1, n + 1):
        total += len(str(i))
    print(total)
elif task == "list":
    values = [(i * 17) % 127 for i in range(64)]
    total = 0
    for _ in range(n):
        for value in values:
            total += value
    print(total)
else:
    print("unknown task")
```

JavaScript (Node.js)

File: sources/node/bench.js

```
function fib(n) {
  if (n <= 1) return n;
  return fib(n - 1) + fib(n - 2);
}
function countPrimes(limit) {
  let count = 0;
  for (let candidate = 2; candidate <= limit; candidate++) {
    let divisor = 2;
    let prime = true;
    while (divisor * divisor <= candidate) {
      if (candidate % divisor === 0) { prime = false; break; }
      divisor++;
    }
    if (prime) count++;
  }
  return count;
}
const task = process.argv[2];
const n = Number(process.argv[3] || 0);
if (task === "hello") {
  console.log("Hello, world!");
} else if (task === "print") {
  for (let i = 1; i <= n; i++) console.log(`line ${i}`);
} else if (task === "sum") {
  let total = 0;
  for (let i = 1; i <= n; i++) total += i;
  console.log(total);
} else if (task === "fib") {
  console.log(fib(n));
} else if (task === "primes") {
  console.log(countPrimes(n));
} else if (task === "format") {
  let total = 0;
  for (let i = 1; i <= n; i++) total += String(i).length;
  console.log(total);
} else if (task === "list") {
  const values = Array.from({length: 64}, (_, i) => (i * 17) % 127);
  let total = 0;
  for (let pass = 0; pass < n; pass++) for (const value of values) total += value;
  console.log(total);
} else {
  console.log("unknown task");
}
```

Ruby

File: sources/ruby/bench.rb

```
def fib(n)
  return n if n <= 1
  fib(n - 1) + fib(n - 2)
end

def count_primes(limit)
  count = 0
  candidate = 2
  while candidate <= limit
    divisor = 2
    prime = true
    while divisor * divisor <= candidate
      if candidate % divisor == 0
        prime = false
        break
      end
      divisor += 1
    end
    count += 1 if prime
    candidate += 1
  end
  count
end
```

```

        prime = false
        break
      end
      divisor += 1
    end
    count += 1 if prime
    candidate += 1
  end
  count
end

task = ARGV[0]
n = (ARGV[1] || "0").to_i
case task
when "hello"
  puts "Hello, world!"
when "print"
  i = 1
  while i <= n
    puts "line #{i}"
    i += 1
  end
when "sum"
  total = 0
  i = 1
  while i <= n
    total += i
    i += 1
  end
  puts total
when "fib"
  puts fib(n)
when "primes"
  puts count_primes(n)
when "format"
  total = 0
  i = 1
  while i <= n
    total += i.to_s.length
    i += 1
  end
  puts total
when "list"
  values = Array.new(64) { |i| (i * 17) % 127 }
  total = 0
  n.times { values.each { |value| total += value } }
  puts total
else
  puts "unknown task"
end

```

Java

File: sources/java/Bench.java

```

import java.util.ArrayList;

public class Bench {
  static long fib(long n) {
    if (n <= 1) return n;
    return fib(n - 1) + fib(n - 2);
  }
  static long countPrimes(long limit) {
    long count = 0;
    for (long candidate = 2; candidate <= limit; candidate++) {

```

```

        long divisor = 2;
        boolean prime = true;
        while (divisor * divisor <= candidate) {
            if (candidate % divisor == 0) { prime = false; break; }
            divisor++;
        }
        if (prime) count++;
    }
    return count;
}

public static void main(String[] args) {
    String task = args[0];
    long n = args.length > 1 ? Long.parseLong(args[1]) : 0;
    switch (task) {
        case "hello": System.out.println("Hello, world!"); break;
        case "print":
            for (long i = 1; i <= n; i++) System.out.println("line " + i);
            break;
        case "sum":
            long total = 0;
            for (long i = 1; i <= n; i++) total += i;
            System.out.println(total);
            break;
        case "fib": System.out.println(fib(n)); break;
        case "primes": System.out.println(countPrimes(n)); break;
        case "format":
            long chars = 0;
            for (long i = 1; i <= n; i++) chars += Long.toString(i).length();
            System.out.println(chars);
            break;
        case "list":
            ArrayList<Long> values = new ArrayList<>();
            for (long i = 0; i < 64; i++) values.add((i * 17) % 127);
            long listTotal = 0;
            for (long pass = 0; pass < n; pass++)
                for (long value : values) listTotal += value;
            System.out.println(listTotal);
            break;
        default: System.out.println("unknown task");
    }
}
}

```

PHP

File: sources/php/bench.php

```

<?php
function fib($n) {
    if ($n <= 1) return $n;
    return fib($n - 1) + fib($n - 2);
}

function count_primes($limit) {
    $count = 0;
    for ($candidate = 2; $candidate <= $limit; $candidate++) {
        $divisor = 2;
        $prime = true;
        while ($divisor * $divisor <= $candidate) {
            if ($candidate % $divisor === 0) { $prime = false; break; }
            $divisor++;
        }
        if ($prime) $count++;
    }
    return $count;
}

$task = $argv[1];

```

```
$n = isset($argv[2]) ? (int)$argv[2] : 0;
if ($task === "hello") {
    echo "Hello, world!\n";
} elseif ($task === "print") {
    for ($i = 1; $i <= $n; $i++) echo "line $i\n";
} elseif ($task === "sum") {
    $total = 0;
    for ($i = 1; $i <= $n; $i++) $total += $i;
    echo $total, "\n";
} elseif ($task === "fib") {
    echo fib($n), "\n";
} elseif ($task === "primes") {
    echo count_primes($n), "\n";
} elseif ($task === "format") {
    $total = 0;
    for ($i = 1; $i <= $n; $i++) $total += strlen((string)$i);
    echo $total, "\n";
} elseif ($task === "list") {
    $values = [];
    for ($i = 0; $i < 64; $i++) $values[] = ($i * 17) % 127;
    $total = 0;
    for ($pass = 0; $pass < $n; $pass++) foreach ($values as $value) $total += $value;
    echo $total, "\n";
} else {
    echo "unknown task\n";
}
```

Go

File: sources/go/bench.go

```
package main

import (
    "bufio"
    "fmt"
    "os"
    "strconv"
)

func fib(n int64) int64 {
    if n <= 1 {
        return n
    }
    return fib(n-1) + fib(n-2)
}

func countPrimes(limit int64) int64 {
    var count int64
    for candidate := int64(2); candidate <= limit; candidate++ {
        prime := true
        for divisor := int64(2); divisor*divisor <= candidate; divisor++ {
            if candidate%divisor == 0 {
                prime = false
                break
            }
        }
        if prime {
            count++
        }
    }
    return count
}

func main() {
    if len(os.Args) < 2 {
```

```

    fmt.Println("unknown task")
    return
}
task := os.Args[1]
var n int64
if len(os.Args) > 2 {
    parsed, err := strconv.ParseInt(os.Args[2], 10, 64)
    if err != nil {
        fmt.Fprintln(os.Stderr, err)
        os.Exit(2)
    }
    n = parsed
}
switch task {
case "hello":
    fmt.Println("Hello, world!")
case "print":
    writer := bufio.NewWriter(os.Stdout)
    for i := int64(1); i <= n; i++ {
        fmt.Fprintf(writer, "line %d\n", i)
    }
    if err := writer.Flush(); err != nil {
        fmt.Fprintln(os.Stderr, err)
        os.Exit(1)
    }
case "sum":
    var total int64
    for i := int64(1); i <= n; i++ {
        total += i
    }
    fmt.Println(total)
case "fib":
    fmt.Println(fib(n))
case "primes":
    fmt.Println(countPrimes(n))
case "format":
    var total int64
    for i := int64(1); i <= n; i++ {
        total += int64(len(strconv.FormatInt(i, 10)))
    }
    fmt.Println(total)
case "list":
    values := make([]int64, 64)
    for i := int64(0); i < int64(len(values)); i++ {
        values[i] = (i * 17) % 127
    }
    var total int64
    for pass := int64(0); pass < n; pass++ {
        for _, value := range values {
            total += value
        }
    }
    fmt.Println(total)
default:
    fmt.Println("unknown task")
}
}

```

Nim

File: sources/nim/bench.nim

```

import os, strutils

proc fib(n: int): int =

```

```
    if n <= 1: return n
    fib(n - 1) + fib(n - 2)

proc countPrimes(limit: int): int =
  var count = 0
  var candidate = 2
  while candidate <= limit:
    var divisor = 2
    var prime = true
    while divisor * divisor <= candidate:
      if candidate mod divisor == 0:
        prime = false
        break
      inc divisor
    if prime: inc count
    inc candidate
  count

if paramCount() < 1:
  echo "unknown task"
  quit(0)

let task = paramStr(1)
var n = 0
if paramCount() > 1: n = parseInt(paramStr(2))

case task
of "hello":
  echo "Hello, world!"
of "print":
  var i = 1
  while i <= n:
    stdout.writeLine("line " & $i)
    inc i
of "sum":
  var total = 0
  var i = 1
  while i <= n:
    total += i
    inc i
  echo total
of "fib":
  echo fib(n)
of "primes":
  echo countPrimes(n)
of "format":
  var total = 0
  var i = 1
  while i <= n:
    total += ($i).len
    inc i

  echo total
of "list":
  var values: array[64, int]
  for i in 0 ..< values.len:
    values[i] = (i * 17) mod 127
  var total = 0
  var pass = 0
  while pass < n:
    for value in values:
      total += value
    inc pass
  echo total
else:
  echo "unknown task"
```

Kotlin (JVM)

File: sources/kotlin/Bench.kt

```
fun fib(n: Long): Long {
    if (n <= 1L) return n
    return fib(n - 1) + fib(n - 2)
}

fun countPrimes(limit: Long): Long {
    var count = 0L
    var candidate = 2L
    while (candidate <= limit) {
        var divisor = 2L
        var prime = true
        while (divisor * divisor <= candidate) {
            if (candidate % divisor == 0L) {
                prime = false
                break
            }
            divisor++
        }
        if (prime) count++
        candidate++
    }
    return count
}

fun main(args: Array<String>) {
    if (args.isEmpty()) {
        println("unknown task")
        return
    }
    val task = args[0]
    val n = if (args.size > 1) args[1].toLong() else 0L
    when (task) {
        "hello" -> println("Hello, world!")
        "print" -> for (i in 1L..n) println("line $i")
        "sum" -> {
            var total = 0L
            for (i in 1L..n) total += i
            println(total)
        }
        "fib" -> println(fib(n))
        "primes" -> println(countPrimes(n))
        "format" -> {
            var total = 0L
            for (i in 1L..n) total += i.toString().length.toLong()
            println(total)
        }
        "list" -> {
            val values = (0L until 64L).map { (it * 17L) % 127L }
            var total = 0L
            var pass = 0L
            while (pass < n) {
                for (value in values) total += value
                pass++
            }
            println(total)
        }
        else -> println("unknown task")
    }
}
```

C/Clang (baseline)

File: sources/c/bench.c

```
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

static int64_t fib(int64_t n) { return n <= 1 ? n : fib(n - 1) + fib(n - 2); }
static int64_t count_primes(int64_t limit) {
    int64_t count = 0;
    for (int64_t candidate = 2; candidate <= limit; candidate++) {
        int64_t divisor = 2;
        int prime = 1;
        while (divisor * divisor <= candidate) {
            if (candidate % divisor == 0) { prime = 0; break; }
            divisor++;
        }
        if (prime) count++;
    }
    return count;
}

int main(int argc, char **argv) {
    const char *task = argv[1];
    int64_t n = argc > 2 ? strtoll(argv[2], NULL, 10) : 0;
    if (strcmp(task, "hello") == 0) puts("Hello, world!");
    else if (strcmp(task, "print") == 0) for (int64_t i = 1; i <= n; i++) printf("line %lld\n", (long long)i);
    else if (strcmp(task, "sum") == 0) {
        int64_t total = 0;
        for (int64_t i = 1; i <= n; i++) total += i;
        printf("%lld\n", (long long)total);
    } else if (strcmp(task, "fib") == 0) printf("%lld\n", (long long)fib(n));
    else if (strcmp(task, "primes") == 0) printf("%lld\n", (long long)count_primes(n));
    else if (strcmp(task, "format") == 0) {
        int64_t total = 0;
        char buffer[32];
        for (int64_t i = 1; i <= n; i++) {
            int length = snprintf(buffer, sizeof(buffer), "%lld", (long long)i);
            total += length;
        }
        printf("%lld\n", (long long)total);
    } else if (strcmp(task, "list") == 0) {
        int64_t values[64];
        for (int64_t i = 0; i < 64; i++) values[i] = (i * 17) % 127;
        int64_t total = 0;
        for (int64_t pass = 0; pass < n; pass++)
            for (int i = 0; i < 64; i++) total += values[i];
        printf("%lld\n", (long long)total);
    } else puts("unknown task");
    return 0;
}
```

R (tidak diukur; Rscript tidak tersedia)

File: sources/r/bench.R

```
args <- commandArgs(trailingOnly=TRUE)
task <- args[[1]]
n <- if (length(args) > 1) as.numeric(args[[2]]) else 0
fib <- function(n) if (n <= 1) n else fib(n - 1) + fib(n - 2)
count_primes <- function(limit) {
    count <- 0
    candidate <- 2
    while (candidate <= limit) {
```

```
divisor <- 2
prime <- TRUE
while (divisor * divisor <= candidate) {
  if (candidate %% divisor == 0) { prime <- FALSE; break }
  divisor <- divisor + 1
}
if (prime) count <- count + 1
candidate <- candidate + 1
}
count
}
if (task == "hello") {
  cat("Hello, world!\n")
} else if (task == "print") {
  i <- 1
  while (i <= n) { cat("line", i, "\n", sep=""); i <- i + 1 }
} else if (task == "sum") {
  total <- 0; i <- 1
  while (i <= n) { total <- total + i; i <- i + 1 }
  cat(format(total, scientific=FALSE, trim=TRUE), "\n", sep="")
} else if (task == "fib") {
  cat(format(fib(n), scientific=FALSE, trim=TRUE), "\n", sep="")
} else if (task == "primes") {
  cat(count_primes(n), "\n", sep="")
} else if (task == "format") {
  total <- 0; i <- 1
  while (i <= n) { total <- total + nchar(as.character(i), type="bytes"); i <- i + 1 }
  cat(total, "\n", sep="")
} else if (task == "list") {
  values <- numeric(64)
  for (i in 0:63) values[i + 1] <- (i * 17) %% 127
  total <- 0; pass <- 0
  while (pass < n) {
    i <- 1
    while (i <= length(values)) { total <- total + values[i]; i <- i + 1 }
    pass <- pass + 1
  }
  cat(total, "\n", sep="")
} else {
  cat("unknown task\n")
}
```