```c
int a[] = {0, 1, 2, 3, 4};

r((int)a, 0, 5);

void r(int a, int i, int E) {
    printf("%d\n", *((int*)a + i));
    if (++i < E) r(a, i, E);
}
```
Learning objectives

CS240 is your introduction to low-level programming

You will learn ...

‣ to solve problems computationally
‣ to design, implement, test, debug and evaluate complex algorithms
‣ about language-level and machine-level representations of control and data
‣ to use production-level tools (C, Unix, Emacs, gdb, make, shell...)

Our programming language of choice is C, because

‣ it is widely used in the industry
  complex systems (from web browsers to operating systems) written in C/C++/Objective-C
‣ it gives you fine-grain control over resources
  whereas Java hides everything from you
‣ it allows you to explore the interaction between software and hardware
  C exposes architectural features
13 homeworks (programming assignments), ~13 take-home quizzes, ~7 in-class quizzes, 2 midterms, 1 final

Grade mix:

- homeworks and quizzes 45%, midterms 30%, final 25%
Piazza

Main forum for interacting with instructors, TAs, and other students

- Link: [www.piazza.com/CS240](http://www.piazza.com/CS240)

Used to ask and clarify any issues pertaining to the course:
- lectures
- labs and projects
Late policy

None.

Rationale: In a large class it is not possible to accommodate requests for extensions

All assignments are handed on time.

- A penalty of 5% per quarter hour will be charged to all assignments submitted after 9:00 pm on the day the assignment is due.
- No assignment will be accepted after 11:59 PM on the due date.
Academic integrity

Any case of cheating will be handled by the Dean of students.
You are encouraged to discuss problems and approaches but:

- Sharing solution is not allowed.
- Buying solutions is not allowed.
- Copying code from the internet is not allowed.
- Copying code from other students is not allowed.
- Copying partial code from other students is not allowed.

http://homes.cerias.purdue.edu/~spaf/cpolicy.html

Each year we catch students... they end up with an F... and a record... is it worth it?
Generative AI

- Useful as an additional resource
- *Not as a substitute student*
- Won’t be able to use them on exams!
Class attendance is mandatory

- If you miss class, get someone else’s notes...

Lab attendance is optional

- If you miss a lab, try to get in on another session during the same week

**Rationale:**

- *Slides may not be complete*
- *To prepare for exams, trust your notes and the book*
Office hours

Lab sessions are your first line of defense
  ▶ Ask as many questions as you can…

Piazza is your second best bet
  ▶ Can ask a question either in public or private
Quizzes

Short tests of your understanding of the lecture material

• Take-home: 24 hours to submit, 1 hour to complete once started

• In-class: 5-7 minutes

Quizzes cover material directly found in the book

It is your responsibility to read the book and ask questions ahead of class in case something is unclear

In-class quizzes will take place in the first five minutes of class and are usually unannounced
Questions

Use the following algorithm

‣ Ask on Piazza
‣ Ask TA at lab
‣ Ask Prof during class

*Rationale: This focuses the interaction and ensure best use of your time*

Regrading questions

‣ Regrading will only be done the week following release of the grade
‣ Contact the TA responsible for the assignment/exam (see syllabus)
‣ Midterm/Final issues are dealt by the Instructor, project/labs are dealt by your TA and can be escalated to the Faculty member

*It is your responsibility to check your grades!*
Questions

How to ask a question on Piazza:
- Read the book, slides, notes
- Describe the problem clearly, using the right terms
- Add output from compiler
- Add any other relevant information

Be polite and respectful of TAs’ time and we’ll do the same

Avoid anonymous questions...
History
Is programming a craft? an art? a science?

▪️ There are many ways to express some task
▪️ How do we know which is best?

We need to understand the tradeoffs…

▪️ e.g. iteration vs. recursion

```c
int a[] = {0, 1, 2, 3, 4};

void r(int* a, int i, int E) {
    for (int i=0; i<5; i++)
        printf("%d\n", a[i]);
    printf("%d\n", *(a+i));
    if (++i < E)
        r(a, i, E);
}
```
int a[]={0,1,2,3,4};  // create a 5 element array

for (int i=0; i<5; i++)
{
    printf("%d\n", a[i]);
}

// loop from zero to five, step by one
// print i-th elem followed by newline
... and in Java

```java
int[] a = new int[]{0,1,2,3,4};
for (int i=0; i<5; i++)
    System.out.println(a[i]);
```

```c
int a[]={0,1,2,3,4};
for (int i=0; i<5; i++)
    printf("%d\n",a[i]);
```
Why C?

Privacy

industry collaboration on that work. We hope you’ll join us.

devise a robust understanding, a common vocabulary, and an explicit set of engineering trade-offs. The second distinction is domain. Are they working in an embedded environment with restricted

The Rust developer community has been the fastest growing over the last two years, but based on historical trends, we

Rust is challenging to learn. Of the more than 8,000 developers responding to the 2020 Rust user survey, only about 100

Once you have gone through the Rust website, you should be comfortable enough to start building things, but there is

post on the user forum or talk directly on the community Discord server. The Discord server is usually the fastest way to

array, we could see the function getting significantly faster by adding concurrency.

When borrowing data, memory doesn’t have to be copied. The memory stays where it is, and a pointer is passed around.

And this gets us to the second innovation that has enabled Rust: the borrow checker. When writing larger programs we

either moving the data or dropping it. By using a

because we are violating the ownership rules. Already, we are seeing how Rust helps us prevent bugs.

some other go routine on the right side, the gift is received and opened. The Go’s garbage collector is going to manage

Again, Rust isn’t the first e

incredible improvements, and because the server is able to respond to requests far more e

on the left and the Rust implementation is on the right. While the GC spike pattern is gone on the Rust graph, the really

CPU and response time graphs when it’s running.

always about performance.

50% performance improvements are great, but here are some other graphs from that migration. Tenable also saw a 75%

Looking again at that study about correlation, we have measurements for more than just energy consumption. The

Renewables should not replace energy e

process, and machine learning training and inference required to support those technologies, it seems unlikely that

contributors

is to support the growth and innovation of Rust, and the member companies have grown from the founding 5 to 27

Energy Efficiency across Programming Languages

How Do Energy, Time, and Memory Relate?

Rui Pereira

University of Minho, Portugal

rapereira@di.uminho.pt

Marco Corino

HSLAB/INESC TEC

University of Minho, Portugal

marco.corino@inesc.pt

Francisco Ribeiro, Rui Raia

HSLAB/INESC TEC

University of Minho, Portugal

fribeiro@di.uminho.pt

madraia@di.uminho.pt

Jácome Cunha

HSLAB/INESC TEC

University of Minho, Portugal

jaacom@c3if.uminho.pt

Júlio Paulo Fernandes

ReleaseLab/CRISC

University of Madeira, Portugal

jpfernandes@crisc.madera.pt

Júlio Saraiva

HSLAB/INESC TEC

University of Minho, Portugal

jsaraiva@di.uminho.pt

Abstract

This paper presents a study of the runtime, memory usage, and energy consumption of twenty-seven well-known soft-

languages. We monitor the performance of each language using two different programming problems, expressed in
each of the languages. Our results show interesting findings, such as slower faster languages consuming less/more

energy, and how memory usage influences energy consumption.

We show how to use our results to provide software engineers support to decide which language to use when energy

efficiency is a concern.


Keywords: Energy Efficiency, Programming Languages, Language Benchmarking, Green Software

ACM Reference Format:


1 Introduction

Software language engineering provides powerful techniques and tools to design, implement and evolve software lan-

guages. Such techniques aim at improving programmers’ productivity by incorporating advanced features in the lan-
guage design, like for instance powerful modules and type systems – and at efficiently execute such software – by de-

veloping, for example, aggressive compiler optimizations. Indeed, most techniques were developed with the main goal

of helping software developers in producing faster programs. In fact, in the last century performance in software languages

was in almost all cases synonymous of fast-execution time (embedded systems were probably the single exception).

In this century, this reality is quickly changing and soft-

ware energy consumption is becoming a key concern for computer manufacturers, software language engineers, pro-

grammers, and even regular computer users. Nowadays, it is used to use mobile phone users (which are powerful com-

puters) avoiding using CPU intensive applications just to save battery/energy. While the concern on the computers’

energy efficiency started by the hardware manufacturers, it quickly became a concern for software developers too [23].

In fact, this is a recent and intensive area of research where several techniques to analyze and optimize the energy con-

sumption of software systems are being developed. Such techniques already provide knowledge on the energy effi-
ciency of data structures [15, 17] and individual language [12], the energy impact of different programming practices both in mobile [13, 22, 19] and desktop application [20, 21], the energy efficiency of applications within the same system [14, 17], or even on how to predict energy consumption in several software systems [4, 5, 13], among several others.

An interesting question that frequently arises in the soft-

ware energy efficiency area is whether a faster program is also an energy efficient program or not. If the answer is yes, then optimizing a program for speed also means optimizing it for energy, and this is exactly what the compiler con-

struction community has been hardly doing since the very beginning of software languages. However, energy consump-

tion does not depend only on execution time, as shown in the equation $E = C + F \times P_{exec}$. In fact, there are

several research works showing different results regarding

What conclusions can we draw from these results?

What are the characteristics of a programming language that influence these characteristics?
Getting started

#include <stdio.h>

int main() {
    printf("Hello World!\n");
}

public class Hello {
    public static void main(String[] s) {
        System.out.println("Hello World!");
    }
}
# Compilation

Start with a human readable file containing your source program and possibly some references to libraries.

```c
#include <stdio.h>
define HELLO "Hello World!\n"
int main() {
    printf(HELLO);
}
```

gcc -std=c99 -c hello.c

gcc -o a.out hello.o

`./a.out`
Compilation

```c
#include <stdio.h>
#define HELLO "Hello World!\n"
int main() {
    printf(HELLO);
}
```

The compiler works in phases that transform the program into the executable one step at a time.

```
 gcc -std=c99 -E hello.c
```

One of the first steps is to expand macro definitions and include external declarations.
#include <stdio.h>
#define HELLO "Hello World!\n"
int main() {
    printf(HELLO);
}

gcc -c hello.c
 gcc -o a.out hello.o
 ./a.out

The compiler works in phases that transform the program into the executable one step at a time.

The last step links libraries, e.g. I/O, to create a stand alone executable binary
Roadmap

Introduction

Language Overview

Datatypes, File I/O

More File I/O

Control-flow, assert()

HW 0

HW 1

HW 2

Structures

Arrays, Memory

More on memory layout

Bitfields, union

HW 3

HW 4

HW 5

Points

More on Pointers

HW 6

HW 7

HW 8

Address-of, malloc

Linked Lists

HW 9

HW 10

HW 11

Recursion

Trees

Bitwise Operations

Types

HW 12

HW 13

Libraries

More on Libraries

HW 14

Midterm

Midterm

Midterm

No Class

No Class

HW 15

HW 16

HW 17

More on Rust

Review

HW 18

HW 19

HW 20

Rust

Buffer Overflow
```c
#include <stdio.h>
int main(void)
{
    int count;
    for (count = 1; count <= 500; count++)
        printf("I will not throw paper airplanes in class.");
    return 0;
}
```