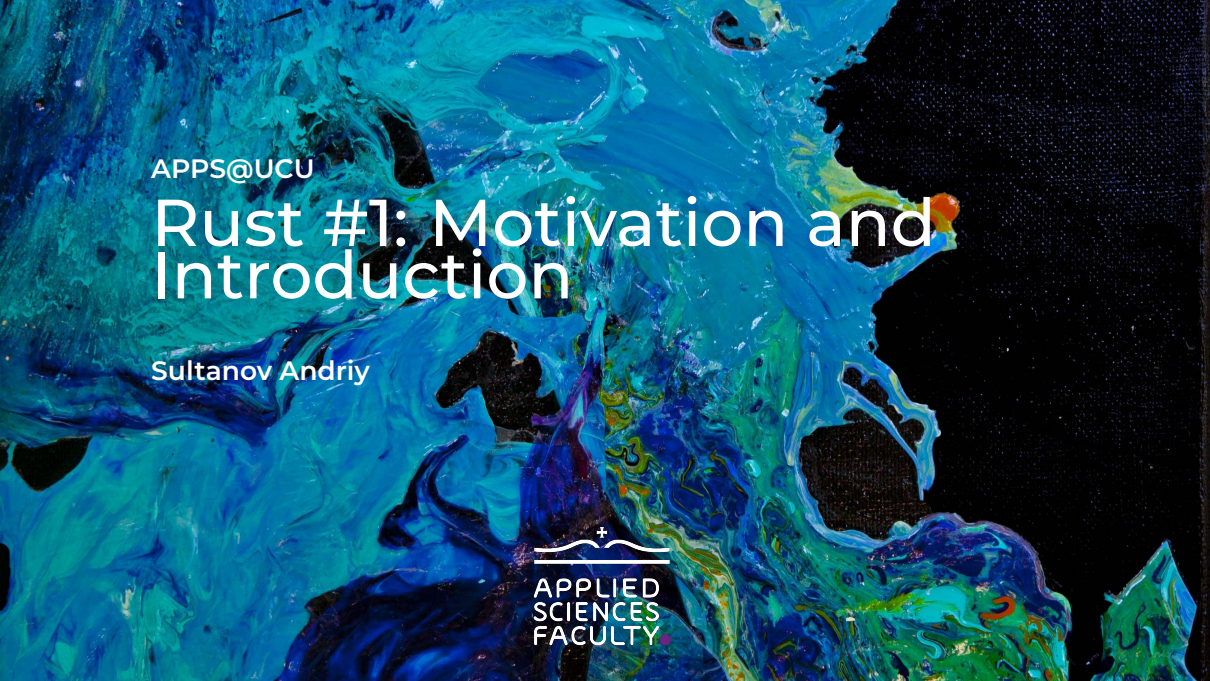

APPS@UCU

Rust #1: Motivation and Introduction

Sultanov Andriy

APPLIED SCIENCES FACULTY

Contents

A short history of systems programming

The origins of C

The C programming language appeared during Unix development in 1972.
Since it was created for a specific purpose and a specific computer , on one hand it adapted to the needs of the programmers, and on the other hand it adopted a large amount of somewhat unique and unpopular ideas and concepts.

Possible solutions

C++ is born to help address some of these problems, introduces ‘zero cost’ abstractions, aimed at providing a nice interface for the programmer to use which compiles down to an almost ideal machine code.
Still has the old instruments and tries to enforce using the new modern safe instruments, which is not ideal. Continues to hang on to C’s model of the machine and programming itself.

Modern ideas

On the other hand, languages like Java, Ruby and Python start sprawling up, presenting another model of growth - they are garbage-collected and are able to present even more complex abstractions without programmers having so much headache, and without any high speed expectations.
Go and others try to address C's problem, unsuccessfully.

Pitfalls of the old ways

General stuff

General stuff about how they are not memory-safe.
Should probably spend some time explaining the memory layout, so it’d be possible to explain buffer overflows, dangling pointers and null pointers.sasasjasa

Buffer overflows

```
int main()
{
        char s[100];
        int i;
        printf("\nEnter a string :  ");
        gets(s);

        return 0;
}
```

Null pointers

```
int test(int* pointer)
{
	*pointer = 5;
	return 0;
}

int main()
{
	char* text = malloc(10);
	*text = 'c';
	return 0;
}
```

Dangling pointers

```
int *test()
{
        int y=10;
        return &y;
}

int main()
{
        int *p = test();
        printf("%d", *p);
        return 0;
}
```

Invalidated iterators

```
void main()
{
	Vec*vec=vec_new();
	vec_push(vec, 107);

	int* n = &vec->data[0];
	vec_push(vec,110);
	printf("%d\n", *n);

	free(vec->data);
	vec_free(vec);
}
```

No real error checking

```
int main()
{
        /*
         * Returns -1 for ERROR #1
         * Returns -2 for ERROR #2
         * .......
         * */
}
```

And many more...

1. Memory leak - you can forget to free data
2. Thread unsafety - another function can be modifying the same memory
3. Double free - you can free the same memory twice (as a part of a struct, for example)

Where and why is Rust better?

What is Rust?

A modern system programming language.
Uses the slogan "Safe, Fast, Easy to write. Choose three"
Gets rid of unnecessary old ideas, combining them with some of the fresh concepts.

Almost everything makes sense

It does not care about C's old ways from the 70s which have been kept up in many languages and systems since just because (null-char strings, pointers) and does not try to needlessly attach new stupid things to it. Easily gets rid of bad ideas since it's a young language.

Memory safety guarantees

Improvements in almost every field

An amazing ecosystem

Basic syntax and concepts

Enumerates and itemizes

The Rust ecosystem and a little more...

cargo