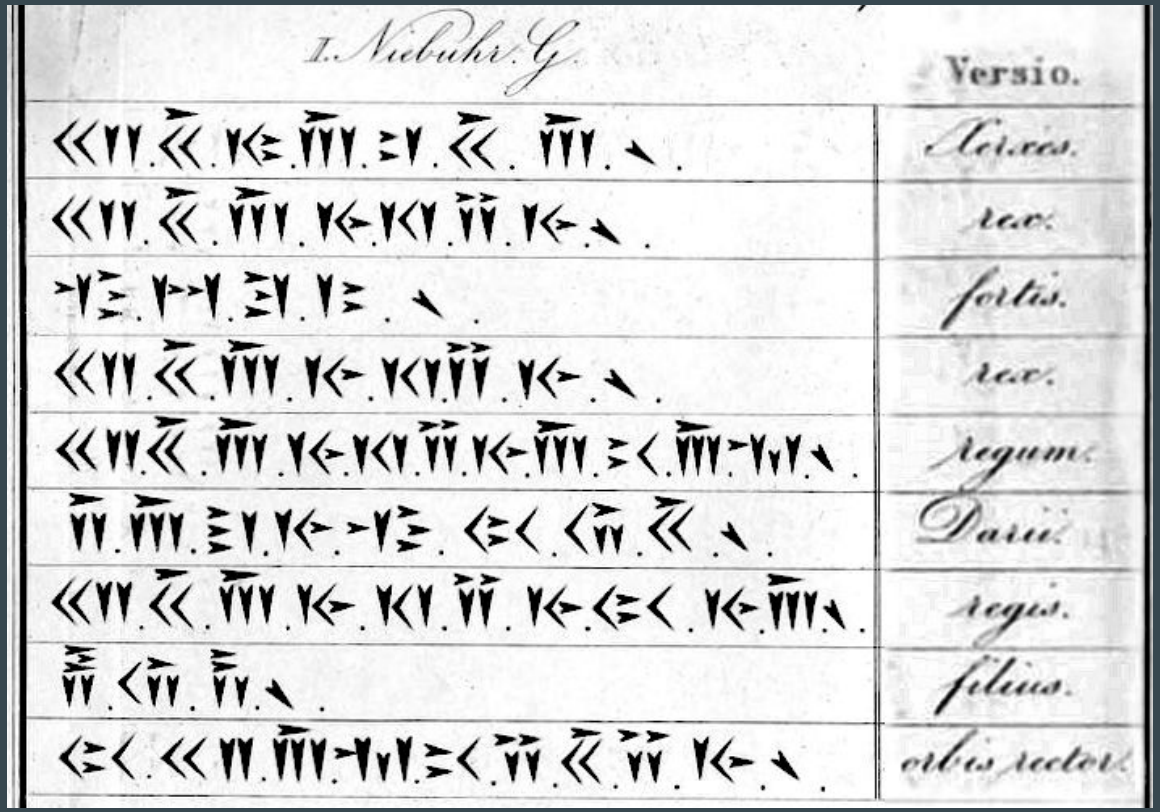
String Theory

...

How hard can it be to deal with text?

Text!


I. Niebuhr. G.
Versio.
Xerxes.
rex.
fortis.
rex.
regum
Darii
regis.
filius.
orbis rector

Text Manipulation in Computer Programs

Different approaches exist for handling textual data.

Goal: understand the tradeoffs of some approaches!

Why? Understand and appreciate design decisions of different programming languages.

C, Java

C “Strings” - Basics

- Everything* is bytes in memory -> Strings are bytes in memory.

```
char *name = "Alice";
```

* Well, yes, but actually, no: some “bytes in memory” have special constraints (read-only, I/O, MMU, …)

C “Strings” - Basics

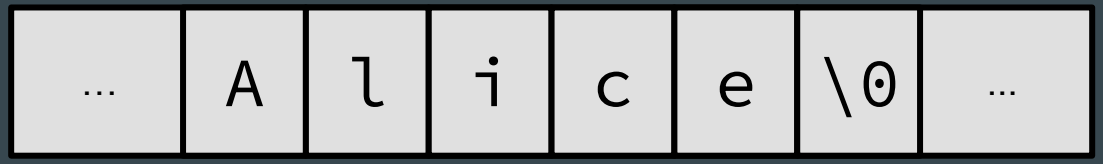
- Everything* is bytes in memory -> Strings are bytes in memory.
- Strings are memory that contains 8-bit chars and ends with ‘\0’

```
char *name = "Alice";
```

C “Strings” - Basics

- Everything* is bytes in memory -> Strings are bytes in memory.
- Strings are memory that contains 8-bit chars and ends with ‘\0’

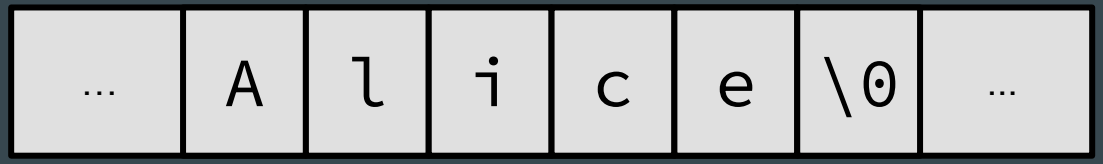
```
char *name = "Alice";
```

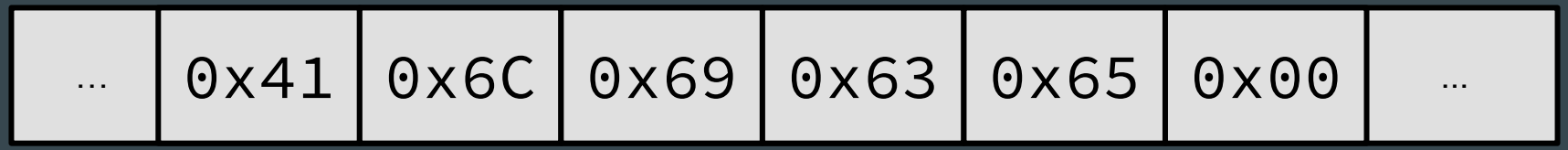


C “Strings” - Basics

- Everything* is bytes in memory -> Strings are bytes in memory.
- Strings are memory that contains 8-bit chars and ends with ‘\0’

```
char *name = "Alice";
```





C “Strings” - Literals

- Everything* is bytes in memory -> Strings are bytes in memory.
- Strings are memory that contains 8-bit chars and ends with `‘\0’`
- `“String literals”`: **constant**, stored in the program binary

```
char *name = "Alice";
name[0] = 'B';
printf("%s", name);
```

C “Strings” - Literals

- Everything* is bytes in memory -> Strings are bytes in memory.
- Strings are memory that contains 8-bit chars and ends with `‘\0’`
- `“String literals”`: **constant**, stored in the program binary (where else?)

C “Strings” - Dynamic strings

- Strings on the heap: arbitrary length, modifiable, `malloc/free`

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

int main() {
    char *name = malloc(12);
    strcpy(name, "Alice");
    printf("%s\n", name);
    name[0] = 'B';
    printf("%s\n", name);
    return 0;
}
```

C “Strings” - Dynamic strings

- Strings on the heap: arbitrary length, modifiable, `malloc/free`
- Helpful tool for any kind of heap allocation or (file) handle work:

  `valgrind`

  - Instruments system such that all allocations are tracked
  - Info about: Memory leaks, invalid read/writes, and more
  - IMHO, highly underrated tool

https://valgrind.org/

C “Strings” - Dynamic strings

- Strings on the heap: arbitrary length, modifiable, `malloc/free`

```
==11286== HEAP SUMMARY:
==11286==     in use at exit: 12 bytes in 1 blocks
==11286==   total heap usage: 2 allocs, 1 frees, 1,036 bytes allocated
==11286==
==11286== LEAK SUMMARY:
==11286==    definitely lost: 12 bytes in 1 blocks
```

C “Strings” - ASCII

Wait.... 8-bit? Isn’t that too small to do anything useful?

- ASCII has 127 letters; gotta be enough (?)
- `wchar_t`: implementation-defined mess
- C11 “`uchar.h`” universal char, ironically not yet supported in many implementations

C “Strings” - ASCII

Wait.... 8-bit? Isn’t that too small to do anything useful?

- ASCII has 127 letters; gotta be enough (?)
- `wchar_t`: implementation-defined mess
- C11 “`uchar.h`” universal char, ironically not yet supported in many implementations

C “Strings” - Things going wrong I

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

int main(int argc, char *argv[]) {
    int size = 0;
    scanf("%d", &size);
    int array[size];

    printf("Size of array: %d\n", sizeof(array));
}
```

C “Strings” - Things going wrong II

```c
#include "stdio.h"

int main() {
    for (int i = 0; i < 10; i++) {
        printf("Hello number \n" + i);
    }
}
```

Java

- Everything* is an object -> Strings are objects
- Strings are immutable - once you got a String, you cannot change it!
    - HOW? The public API is read-only. String is also marked as `‘final’`, preventing overriding

  That’s actually nice: no side-effects with shared Strings, Thread Safety

- StringBuilder or StringBuffer for when you want a buffer of char

* Well, yes, but actually, no: double, char, and friends are considered “primitive” types, on the stack

Java

Unicode is supported, but clumsy

- char is 16-bit on every architecture. 2^16 = 65535 (0xffff)

  But Unicode has 136,755 characters, at least!

- Solution: Store Unicode code point in 2 java chars (UTF-16).
  - Length of String != Number of chars in the String
  - Indexing a Unicode String is potentially an invalid operation

  - `String java = 💩; // works, 0x1F4A9 > 0xffff`

http://wiki.juneday.se/mediawiki/images/b/b4/Java_and_Unicode.pdf

Java

```
String str1 = "someString";
String str2 = "someString";
Scanner in = new Scanner(System.in);

while (true) {
    if (str1 == str2) {
        // Ironically, operator+ for string concatenation exists
        System.out.println("\"someString\" == \"" + str2 + "\" is true!");
    } else {
        System.out.println("\"someString\" == \"" + str2 + "\" is false!");
    }
    str2 = in.nextLine();
}
```

Java

```
String str1 = "someString";
String str2 = "someString";
Scanner in = new Scanner(System.in);

while (true) {
    if (str1 == str2) {
        [illegible] In[illegible]ically [illegible]or [illegible] for [illegible]ing [illegible]ation exists
        [illegible]tem.out.println("\"someString\" == \"" + str[illegible] "\" i[illegible]!");
    } else {
        System.out.println("\"someString\" == \"" + str2 + "\" is false!");
    }
    str2 = in.nextLine();
}
```

Leaky Abstraction

Java

Java does not support operator overloading (all proposals were rejected)

- Operator overloading? -> Assigning new meaning to +, -, !, ==, and so on

  matrix1 + matrix2 becomes something like:
  `Matrix::operator::add(matrix1, matrix2) -> Matrix`

But String is special case; ‘+’ is allowed for string concatenation

- string concatenation? Putting different strings together.

```
System.out.println(“This will be ” + 1 + “ string”);
```

Java

BTW:

Yes, this is how you print a string to Standard Output in Java:

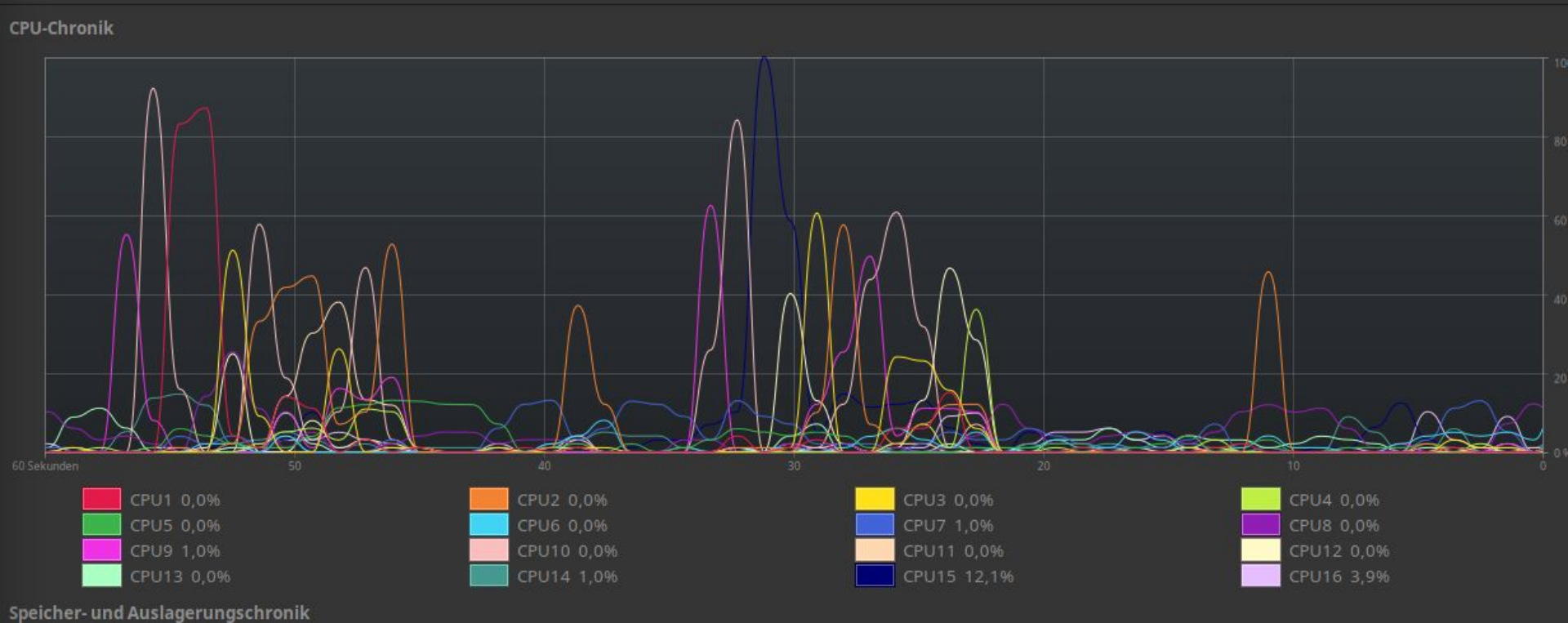
```
System.out.println(“This will be ” + 1 + “ string”);
```

Java

Another operator: ‘==’

Java

```
$> valgrind java StringExample
```



C

https://stackoverflow.com/questions/16870485/how-can-i-read-an-input-string-of-unknown-length