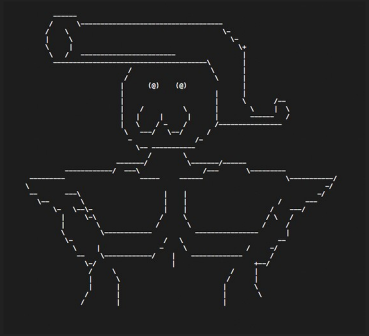
Team “Who Knows Ada” Presents…

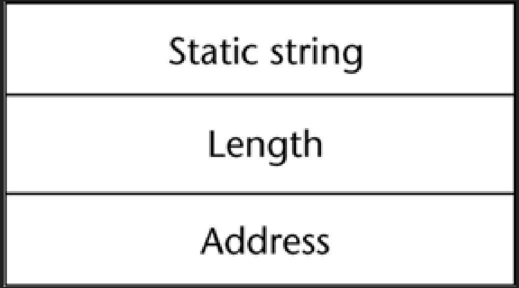
Dynamic & Static Length Strings in C, C#, Java, Scala

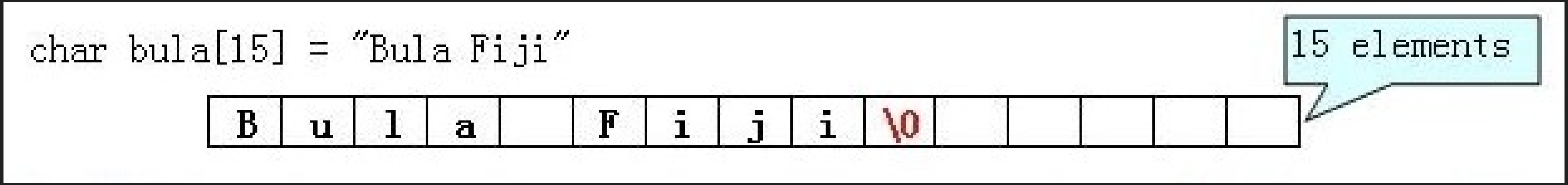
Nick Calkins | Erin Chon | Alan Huang | Harvey Lin



What is a Static Length String?

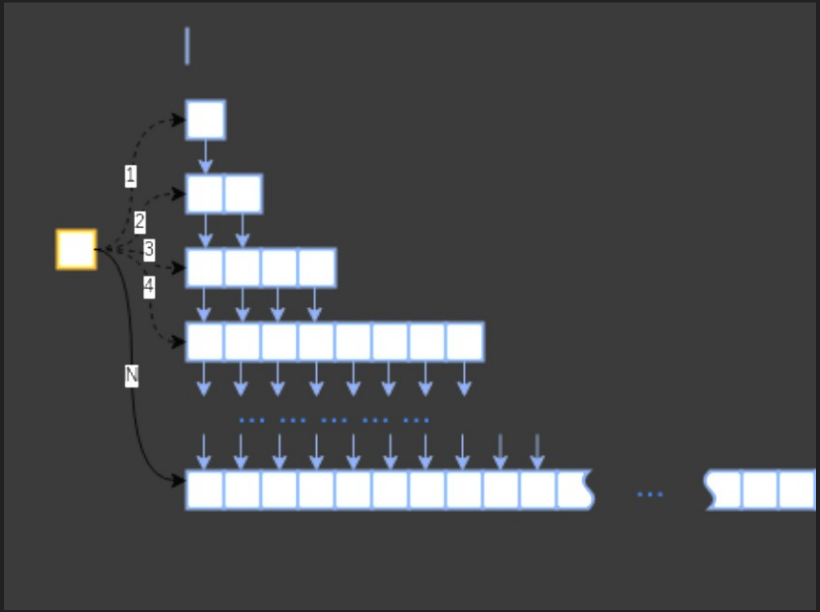
- A string with a fixed allocated length and capacity
  - Can be mutable or immutable





What is a Dynamic Length String?

- A string that may or may not have a fixed capacity but has an allocated length that can change at run-time



Direct Comparison of Static Length vs Dynamic Length Strings

- Pros for Static length:
    - Can be made immutable, which is good for security purposes
    - Allocation of memory is simpler due to not having to account for expansion.
- Cons:
    - Not as flexible as dynamic strings
    - If mutable, must be sufficiently long to accommodate inputs of certain lengths.

- Pros for Dynamic Length:
    - Gives the user flexibility to input a String as long as they want, language handles expansion.
    - No arbitrary limit to length of string
- Cons:
    - Allocation and deallocation is complicated by ability to grow
    - Ability to grow must be coded in somehow.
    - No limit to growth means unwieldy, large strings.
    - Mutability raises security concerns.

Strings in Java (and Scala)

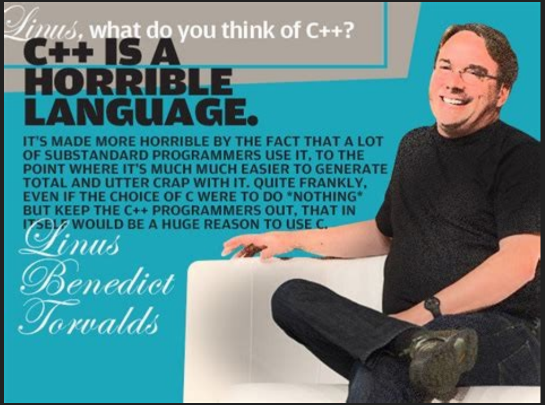
- String objects in Java and Scala are immutable and statically lengthed, but StringBuilder allows for a basis to build these types of strings mutably.

- Static Length: StringBuilder is declared, and the capacity is set to MAXLENGTH, which will create a StringBuilder that allocates a value array for MAXLENGTH characters.
    - User checking is needed to prevent expansion of the stringbuilder beyond this maximum capacity.

- Dynamic Length: StringBuilder is used, and is allowed to grow until the max size of a StringBuilder.

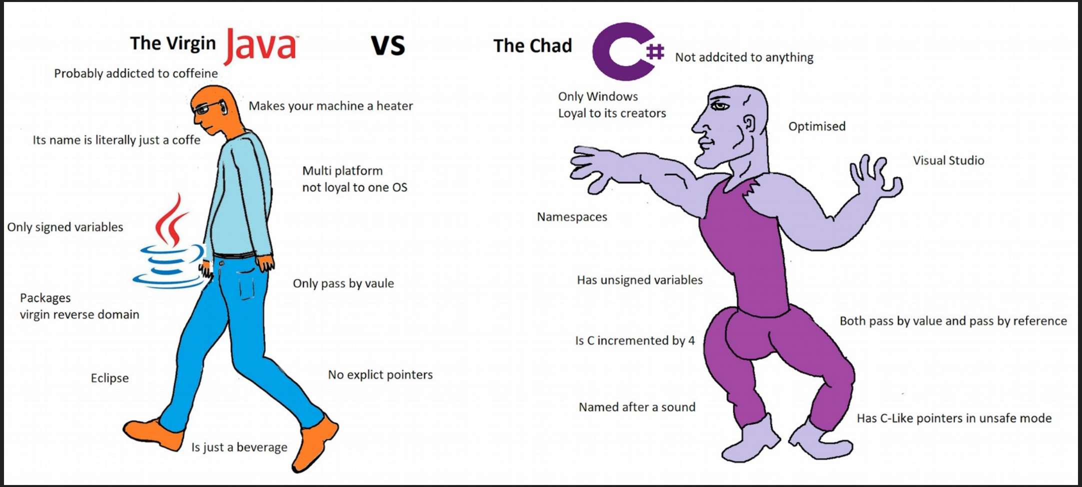
Strings in C

- No dedicated string type (influence from B)
    - All provided types are either literals or pointers
- Strings are represented as char arrays terminated with '\0'
- Common string operations are provided through string.h
- Static length strings are simply static length char arrays and the user performs bounds checking
    - Can be stack allocated, heap allocated, or even statically allocated in the case of string literals
- Dynamic length strings can be easily implemented using malloc and realloc
    - Must be heap allocated


Linus, what do you think of C++?
C++ IS A HORRIBLE LANGUAGE.
IT'S MADE MORE HORRIBLE BY THE FACT THAT A LOT OF SUBSTANDARD PROGRAMMERS USE IT, TO THE POINT WHERE IT'S MUCH MUCH EASIER TO GENERATE TOTAL AND UTTER CRAP WITH IT. QUITE FRANKLY, EVEN IF THE CHOICE OF C WERE TO DO *NOTHING* BUT KEEP THE C++ PROGRAMMERS OUT, THAT IN ITSELF WOULD BE A HUGE REASON TO USE C.
Linus Benedict Torvalds

Strings in C#

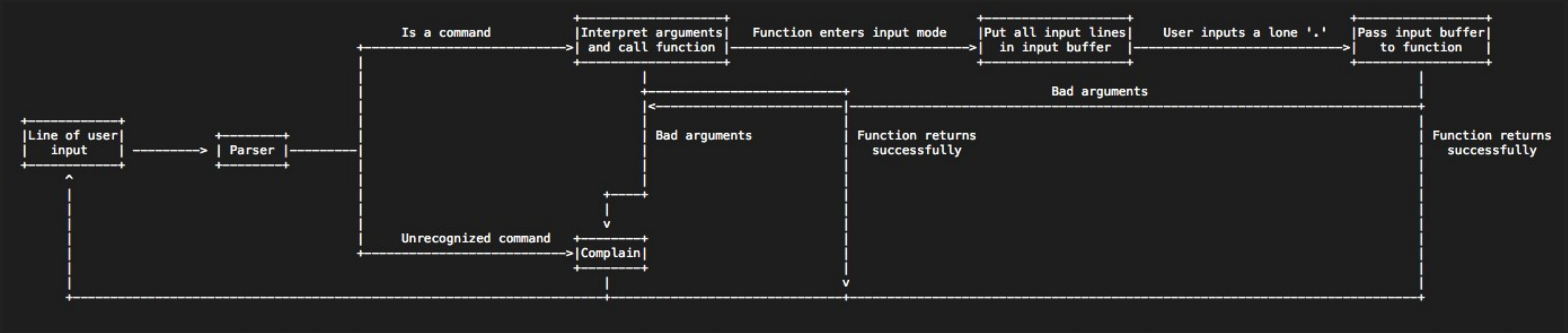
- Java but better. Immutable static length strings provided by String class
- StringBuilder class can directly implement both dynamic and mutable static length strings using its various overloaded constructors
- Like Java, objects are always allocated from the heap and so strings implemented this way are always heap-allocated



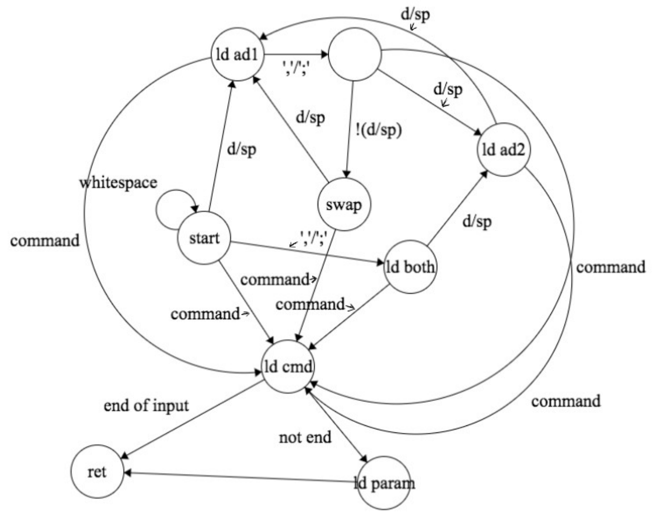
The Project

- Partial implementations of the line text editor ed in all 4 languages using mutable static length and dynamic length strings to measure design and performance impacts
- Supports the commands a, c, d, j, l, m, n, p, t, w, and e
- Command and line addressing syntax mirrors that of GNU/ed
- Program consists of a command parser and text buffers
- Text buffers are implemented as double linked lists with utility functions to support the chosen commands
- Designed so the internal representation of strings can be easily swapped between mutable static and dynamic strings

Structure


Line of user input
Parser
Is a command
Interpret arguments and call function
Function enters input mode
Put all input lines in input buffer
User inputs a lone '.'
Pass input buffer to function
Bad arguments
Bad arguments
Function returns successfully
Function returns successfully
Unrecognized command
Complain

Parser



Parses the format [address[(,/;)address]]command[parameter]

ld = load; ad = address; cmd = command; param = parameter

load the next character on each transition

d = character is a numerical digit

sp = character represents a special address value (excluding ',' and ';')

Demo

C Implementation of ed by Harvey

Conclusions

- Using static length strings provided a small performance boost over dynamic length strings as it avoided the overhead of string resizing and allocation but this varied across languages.
    - ~2% performance increase in the C implementation with the initial dynamic length set to 10
    - In C#, static length StringBuilders resulted in ~10% decrease in performance. This is likely due to inefficient parameter passing by the author however.
    - Scala saw a ~7% decrease but it is likely due to the cost of performing length checks
    - Performance benefit is small for C but lost/inconclusive for higher level languages
- Memory use is notably higher with static length strings as strings are always allocated to their maximum length but the magnitude varies depending on preallocated lengths
    - With a static string length of 1024 and an initial dynamic string length of 8 loading ~130,000 lines from the bible in the C implementation
        - Static string RAM usage = 189.1MB
        - Dynamic string RAM usage = 41.2MB
    - The same test done in the C# implementation
        - 344MB vs. 96MB
    - Same test but with static length of 512 in Scala
        - 124.5MB vs. 85.5MB
- Design impact for the programmer is minimal. For static length strings, checks have to be performed to guard against overflow.
- Reliability impact is significant. Static length strings run the risk of overflowing. For text editing, this trade-off is not worthwhile

References

[1] https://docs.oracle.com/javase/7/docs/api/java/lang/String.html
[2] https://www.gnu.org/software/ed/manual/ed_manual.html
[3] http://hg.openjdk.java.net/jdk8/jdk8/jdk/file/687fd7c7986d/src/share/classes/java/lang/AbstractStringBuilder.java
[4]https://www.scala-lang.org/api/current/scala/Predef$.html
[5]https://www.artima.com/intv/gosling3.html
[6]https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/strings/
[7]http://www.bell-labs.com/usr/dmr/www/chist.html
[8]https://en.cppreference.com/w/cpp/language/string_literal
[9]https://docs.oracle.com/javase/tutorial/java/data/buffers.html
[10]http://cs.boisestate.edu/~alark/cs354/lectures/data_types_chars_strings.pdf