Mathematical String Notation

String Theory

- A mathematical model that we will use often is that of ***mathematical strings***
- A string can be thought of as a series of zero of more ***entries*** of *any* other mathematical type, say, T
  - T is called the ***entry type***
  - We will call this math type **`string of`** T

`String` ≠ ***`string`***

- `String` is a programming type in Java, and ***`string`*** is a mathematical type (often used to model program types)
- Since we call the mathematical model of the Java primitive type **`char`** by the name ***`character`***, we have:

  `type` *`String`* ***`is modeled by`***
  `string of character`

Math Notation for Strings

- The following notations are used when we write mathematics (e.g., in contract specifications) involving strings
- Notice two important features of strings:
  - There may be ***duplicate*** entries (in fact, there may be arbitrarily many of a given entry value)
  - The ***order*** of the entries is important

The Empty String

- The ***empty string***, a string with no entries at all, is denoted by < > or by **`empty_string`**

Denoting a Specific String

- A particular string can be described by listing its entries between `<` and `>` separated by commas
- Examples:

```
< 1, 2, 3, 2 >
< 'G', 'o' >
< >
```

Denoting a Specific String

- A particular str
listing its entri
separated by com
- Examples:

  `< 1, 2, 3, 2 >`

  `< 'G', 'o' >`

  `< >`

A ***string of integer*** value whose entries are the ***integer*** values *1*, *2*, *3*, and *2*.

Denoting a Specific String

- A particular st
  listing its entri
  separated by com
- Examples:

```
< 1, 2, 3, 2 >
< 'G', 'o' >
< >
```

A ***string of character*** value whose entries are the ***character*** values *'G'* and *'o'*.

Denoting a Specific String

- A particular str
  listing its entrie
  separated by comm
- Examples:

```
< 1, 2, 3, 2 >
< 'G', 'o' >
< >
```

We may also use the special notation *"Go"* for the same ***string of character*** value.

Denoting a Specific String

- A particular str
listing its entrie
separated by com
- Examples:

```
< 1, 2,      2 >
< 'G'    'o' >
< >
```

Now it can be seen that this notation for ***empty_string*** is a special case of the string literal notation.

Concatenation

- The ***concatenation*** of strings `s` and `t`, a string consisting of the entries of `s` followed by the entries of `t`, is denoted by `s * t`
- Examples:

  `< 1, 2 > * < 3, 2 > = < 1, 2, 3, 2 >`

  `< 'G', 'o' > * < > = < 'G', 'o' >`

  `< > * < 5, 2, 13 > = <5, 2, 13 >`

  `< > * < > = < >`

Concatenation

- The ***concaten*** string consisti by the entries

As before, we may use the special notation for a ***string of character*** value and say:

"Go" * "" = "Go"

- Examples:

< 1, 2 > * < 2 > = < 1, 2, 3, 2 >

< 'G', 'o' > * < > = < 'G', 'o' >

< > * < 5, 2, 13 > = <5, 2, 13 >

< > * < > = < >

Concatenation

- The ***concatenation*** of strings `s` and `t`, a string consisting of the entries of `s` followed by the entries of `t`, is denoted by `s * t`
- Examples:

The concatenation of Java `String` values uses `+` instead!

= < 1, 2, 3, 2 >

< 'G', 'o' >

<5, 2, 13 >

< > * < > = < >

Substring, Prefix, Suffix

- We say `s` ***`is substring of`*** `t` iff the entries of `s` appear consecutively in `t`
- We say `s` ***`is prefix of`*** `t` iff the entries of `s` appear consecutively at the beginning of `t`
- We say `s` ***`is suffix of`*** `t` iff the entries of `s` appear consecutively at the end of `t`

Substring, Prefix, Suffix

- We say `s` ***`is s`*** … entries of `s` appe…
- We say `s` ***`is p`*** … s of `s` appear consecutively at the beginning of `t`
- We say `s` ***`is suffix of`*** `t` iff the entries of `s` appear consecutively at the end of `t`

> We say ***`is not ...`*** for the negation of each.

Length

- The ***length*** of a string `s`, i.e., the number of entries in `s`, is denoted by `|s|`
- Examples:

```
|< 1, 2, 3, 2 >| = 4
|< 'G', 'o' >| = 2
|< >| = 0
```

Concise Notation for Substrings

- The substring of `s` starting at ***position*** `i` (inclusive) and ending at ***position*** `j` (exclusive) is denoted by `s[i, j)`

Concise Notation for Substrings

- The substring of `s` starting at ***position*** `i` (inclusive) and ending at ***position*** `j` (exclusive) is denoted by `s[i, j)`

The ***position*** `k` of an entry in a string is a number satisfying `0 <= k < |s|`.

Concise Notation for Substrings

- The substring of `s` starting at ***position*** `i` (inclusive) and ending at ***position*** `j` (exclusive) is denoted by `s[i, j)`

This notation is well-defined whenever
`0 <= i <= j <= |s|`;
for all other cases, the designated substring may be defined to be
`<>`
(though we will avoid using this).

Concise Notation for Substrings

- The substring of `s` starting at ***position*** `i` (inclusive) and ending at ***position*** `j` (exclusive) is denoted by `s[i, j)`
- Examples with `s = "GoBucks"`:

  `s[0, |s|) = "GoBucks"`

  `s[2, |s|-1) = "Buck"`

  `s[1, 1) = ""`

  `s[2, 3) * s[5, 7) = "Bks"`

Reverse

- The ***reverse*** of a string `s`, i.e., the string with the same entries as `s` but in the opposite order, is denoted by `rev(s)`
- Examples:

  `rev(< 1, 2, 3, 2 >) = < 2, 3, 2, 1 >`

  `rev(< 'G', 'o' >) = < 'o', 'G' >`

  `rev(< >) = < >`

Permutations

- The question whether strings *s1* and *s2* are ***permutations***, i.e., whether they are simply reorderings of one another, is denoted by ***perms****(s1, s2)*
- Examples:

  perms*(< 1, 2, 3 >, < 3, 1, 2 >)*

  not perms*(< 2, 2, 1 >, < 2, 1 >)*

  perms*(< >, < >)*

Occurence Count

- The ***occurence count*** of an entry *x* in a string *s*, i.e., the number of times *x* appears as an entry in *s*, is denoted by **`count`**`(s, x)`
- Examples:

  `count``(< 2, 2, 2, 1 >, 2) = 3`

  `count``(< 2, 2, 2, 1 >, 4) = 0`

  `count``(< 'G', 'o' >, 'G') = 1`

  `count``(< >, 13) = 0`