

Stint

Jiang Wu, Ningning Xia, Sichang Li, Tingting Ai, Yiming Xu

String and Integer Language

Outline

- Motivation
- Language Features
- Tutorial with Scenario
- Structure Design
- Summary
- Lessons

Background and Motivation

- Strings are mixtures of character blocks and numbers
- Usually we want to manipulate different parts of a string according to what they are
- Existing languages are complicated or heavy-weighted

That's what we focus on!

Language Features

- * Flexible Type Conversion
- * More Convenient String Operations

Language Features

- * Flexible Type Conversion
- * More Convenient String Operations

Auto-Conversion:

string str1 = 12; -> "12"

string str2 = true; -> "true"

Language Features

- * Flexible Type Conversion
- * More Convenient String Operations

Int-Extraction:

"age 22, grade 98, rank 3"

Language Features

- * Flexible Type Conversion
- * More Convenient String Operations

Int-Extraction:

"age **22**, grade **98**, rank **3**"



str .<| 0 |> = str .<| 0 |> + 1;

Language Features

- * Flexible Type Conversion
- * More Convenient String Operations

Insert & Delete:

"this is "

str = str + "Stint";

Language Features

- * Flexible Type Conversion
- * More Convenient String Operations

Insert & Delete:

"this is **Stint**"
 ↑

```
str = str + "Stint";  
str = str + "not " @ 8;
```

Language Features

- * Flexible Type Conversion
- * More Convenient String Operations

Insert & Delete:

"this is **not** Stint"

```
str = str + "Stint";  
str = str + "not " @ 8;
```

Language Features

- * Flexible Type Conversion
- * More Convenient String Operations

Insert & Delete:

"this is not Stint"

str = str - "is";

str = str - "is" @ 4;

"this is not Stint"

"this is not Stint"



Language Features

- * Flexible Type Conversion
- * More Convenient String Operations

Extraction:

"abc123def"

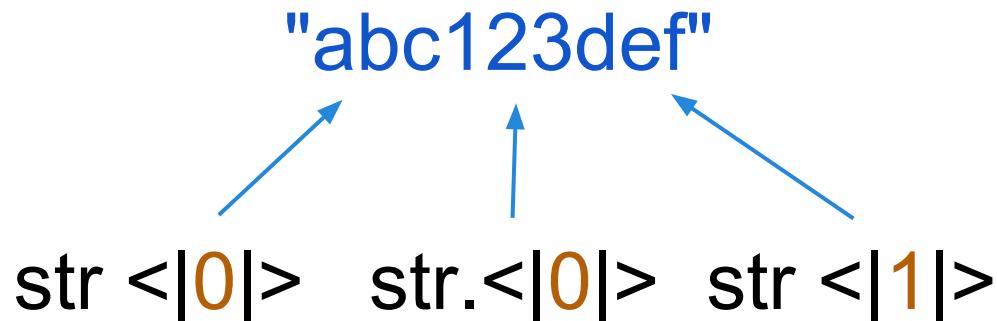
str[0] → "a"

str[0, 3] → "abc"

Language Features

- * Flexible Type Conversion
- * More Convenient String Operations

Extraction:



Language Features

- * Flexible Type Conversion
- * More Convenient String Operations

More Extraction:

The diagram illustrates a string extraction operation. At the top, the string "This is his thesis" is displayed in black font. Overlaid on this string are four red rectangular boxes, each containing one word: "This", "is", "his", and "thesis". Below the string, the indices 0, 1, 2, and 3 are shown in red, corresponding to the start of each word respectively.

```
str | " ";
str <|2|>    -> "his"
```

Language Features

- * Flexible Type Conversion
- * More Convenient String Operations

More Extraction:

"This is thisness"

0 1 2 3

str # "is";
str <|1|>

The diagram illustrates a string extraction operation. At the top, the string "This is thisness" is shown in blue, with its characters indexed from 0 to 3 below it. A blue arrow points from the label "str <|1|>" at the bottom left to the character 'i' in the word "is". This indicates that the substring "is" is being extracted, starting at index 1 and ending at index 2.

Language Features

- * Flexible Type Conversion
- * More Convenient String Operations

Assignment:

"This isn't this is"

0 11 2 2 3 3

str # "is";

str <|1|> = "isn't"

Tutorial with Scenario

Update inventory record of a store:

sn1_table.txt :

```
<name><price><sales><inventory>
MacBookPro $1200 248 203
MacBookAir $900 381 246
iMac $1600 64 98
....
```

sn1_input.txt :

```
<sales in this month>
194
246
63
....
```



sn1_output.txt :

```
<name><mark><price><sales><inventory><income>
MacBookPro(BEST) $1200 442 9 530400
MacBookAir(BEST) $900 627 0 564300
iMac $1600 127 35 203200
....
```



Tutorial with Scenario

```
void main () {
    string input_file = "sn1_table.txt";
    ..... /* save all file names as variables */
    string in_buff = "";
    string data_buff = "";
    int sales;

    /* open input and output files */
    open input_file; ......

    /* read line by line */
    while (input_file >> in_buff) {
        data_file >> data_buff;
        sales = data_buff.<|0|>;
        /* calculate total sale income */
        in_buff = in_buff + " " + (in_buff.<|0|> * in_buff.<|1|>);

        /* mark tag if necessary */
        in_buff | " ";
        if (in_buff.<|3|> > 300000) {
            in_buff<|0|> = in_buff<|0|> + "(BEST)";
            if (in_buff.<|3|> < 100000) {
                in_buff<|0|> = in_buff<|0|> + "(OFFER)";
                in_buff.<|0|> = in_buff.<|0|> / 2;
            }
            output_file << in_buff + "\n";
        }
        close input_file; ..... /* close all opened files */

    return; /* must have a return statement */
}
```

Tutorial with Scenario

```
void main () {
    string input_file = "sn1_table.txt";
    ..... /* save all file names as variables */
    string in_buff = "";
    string data_buff = "";
    int sales;

    /* open input and output files */
    open input_file; .....
    sales = data_buff.<|0|>, .....
```

/* modify sales and inventory */
in_buff.<|1|> = in_buff.<|1|> + sales;
in_buff.<|2|> = in_buff.<|2|> - sales;

```
ulate total sale income */
f = in_buff + " " + (in_buff.<|0|> * in_buff.<|1|>);

<name><price><sales><inventory>
MacBookPro $1200 248 203
MacBookAir $900 381 246
iMac $1600 64 98
.....
in_buff.<|0|> = in_buff.<|0|> + "(OFFER)";
in_buff.<|0|> = in_buff.<|0|> / 2;
```

<sales in this month>
194
246
63
.....

```
close input_file; ...
return; /* must have a return statement */
}
```

Tutorial with Scenario

```
void main () {
    string input_file = "sn1_table.txt";
    ..... /* save all file names as variables */
    string in_buff = "";
    string data_buff = "";
    int sales;

    /* read line by line */
    while (input_file >> in_buff) {
        data_file >> data_buff;
        sales = data_buff.<|0|>;
        /* calculate total sale income */
        in_buff = in_buff + " " + (in_buff.<|0|> * in_buff.<|1|>);

        /* mark tag if necessary */
        in_buff | " ";
        if (in_buff.<|3|> > 300000) {
            in_buff.<|0|> = in_buff.<|0|> + "(BEST)";
        }
        /* output to file */
        output_file << in_buff + "\n";
    }

    /* modify sales and inventory */
    in_buff.<|1|> = in_buff.<|1|> + sales;
    in_buff.<|2|> = in_buff.<|2|> - sales;
}
```

The diagram illustrates the state of variables during the execution of the program. A blue box highlights the value of `data_buff` as `194`. Another blue box highlights the value of `in_buff` as `MacBookPro $1200 248 203`, with a red arrow pointing to the value `203`.

Tutorial with Scenario

```
void main () {  
    string input_file = "sn1_table.txt";  
    ..... /* save all file names as variables */  
    string in_buff = "";  
    string data_buff = "";  
    int sales;
```

in_buff:
MacBookPro \$1200 248 203

MacBookPro \$1200 442 9 530400

MacBookPro(BEST) \$1200 442 9
530400

```
/* calculate total sale income */  
in_buff = in_buff + " " + (in_buff.<|0|> * in_buff.  
<|1|>);  
  
/* mark tag if necessary */  
in_buff | " ";  
if (in_buff.<|3|> > 300000) {  
    in_buff<|0|> = in_buff<|0|> + "(BEST)"; }  
if (in_buff.<|3|> < 100000) {  
    in_buff<|0|> = in_buff<|0|> + "(OFFER)";  
    in_buff.<|0|> = in_buff.<|0|> / 2;  
}  
output_file << in_buff + "\n";  
}
```

Tutorial with Scenario

```
void main() {
    <name><price><sales><inventory>
    MacBookPro $1200 248 203
    MacBookAir $900 381 246
    iMac $1600 64 98
    .....
}

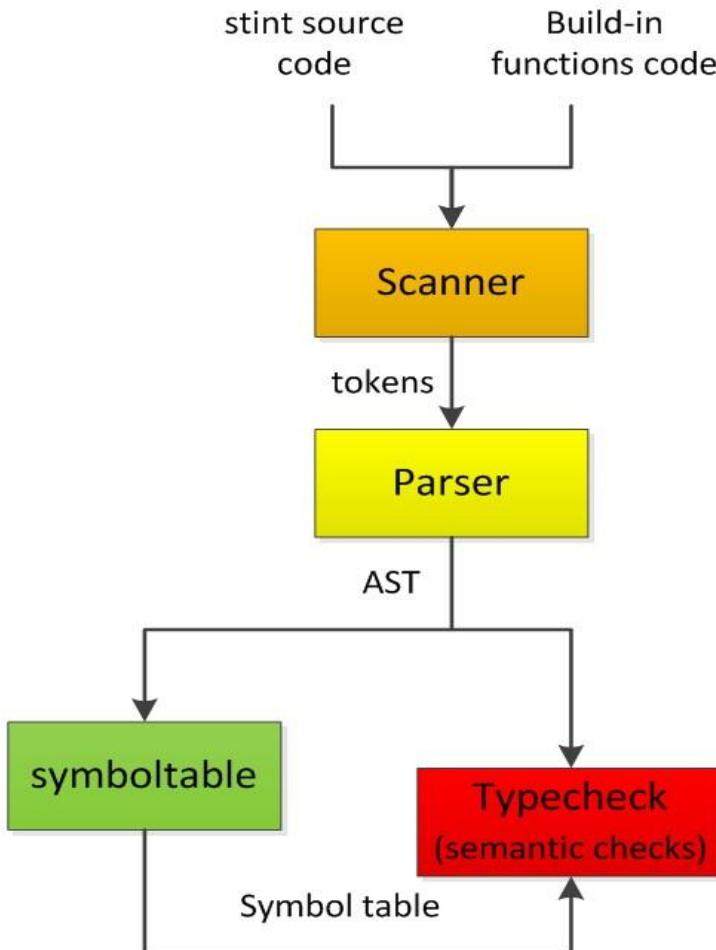
/* open input and output files */
open input_file; .....
/* read input file line by line */
while (input_file >> in_buff) {
    data_file >> data_buff;
    .....
    sales = data_buff.<|0|>,
    /* calculate total sale income */
    in_buff = in_buff + " " + (in_buff.<|0|> * in_buff.<|1|>);

    /* mark tag if necessary */
    in_buff | " ";
    if (in_buff.<|3|> > 300000) {
        in_buff.<|0|> = in_buff.<|0|> + "(BEST)" - 1;
    }
}

<name><mark><price><sales><inventory><income>
MacBookPro(BEST) $1200 442 9 530400
MacBookAir(BEST) $900 627 0 564300
iMac $1600 127 35 203200
.....
close input_file; ..... /* close all opened files */
return; /* must have a return statement */
}
```

Structure Design

Stint compiler structure



6 compilation steps

- scanner--takes the source program and generates a list of tokens.
- parser--takes the tokens and generates the AST.
- symboltable--construct symbol table for locals, globals and functions.
- typecheck--checks the consistency of the AST and returns a new SAST.
- Stint.java--Java helper functions used in the translator.
- translator--translates the SAST to Java code, using the Java helper functions in Stint.java

Finally, to run the program, we use javac to compile the java code to java bytecode, and then we use java to run it.

Summary

- **Goals achieved**

- Simplicity: Use operators to provide basic string operations
- Convenience: Implementation of I/O functions and type conversion
- A strongly typed language

- **Challenges**

- Unfamiliarity with formal grammar
- Translation from Ocaml to Java
- Data structure design

Lessons & Tools

- **Lessons**

- Planning and scheduling
- Thinking and working in functional language
- Team cooperation -share ideas -communicate

- **Tools**

- Ocaml
- Eclipse
- Git - Version Control
- Google Doc - Documentation

Q&A

Thanks