Project Proposal

Locke Meyer, Felix King

0.1 Introduction

KnitPick#++ turns the tedious process of creating a knitting pattern, a set of instructions coupled with a chart that together describe the construction process of a specific garment, into a precise, painless, and intuitive programming experience. Ordinarily, creating a knitting pattern requires both some math skills as well as the graphic design savvy to draw a clear chart; with KnitPick#++, these are no longer required. Furthermore, the various styles of knitting pattern that exist can be confusing to a knitter without a wide range of prior knitting experience; KnitPick#++ establishes a powerful and expressive standard for knitting patterns that can be easily learned and read by anyone.

Many knitting patterns rely not only on repetition of single stitches, but also entire rows of stitches and of sets of rows. Also, since a secondary goal of knitting patterns is for a given pattern to produce the exact same garment every time, the output is deterministic. For these reasons, the problem of generating knitting patterns is particularly suited to a programming language-based solution, since programming languages offer easy repetition of pieces of information as well as precision.

0.2 Design Principles

One of our technical goals for KnitPick#++ is for it to be relatively lightweight in terms of (e.g.) the number of basic stitches that are defined by the language. However, we also want to design KnitPick#++ such that users can grow the language to fit their needs – for instance, by defining a less common stitch that serves a more niche purpose in the knitting process. Another one of our goals for KnitPick#++ is to have it generate documents that look nice and are easy to comprehend. One way that this might inform the design of the language: rather than packing all possible information about a given pattern into a document for output, the user will be able to generate documents with varying amounts of detail based on their specifications.

0.3 Examples

A.

```
pg Body =
    row stock1 = s1 k+
    row stock2 = s1 p+
    repeat 75 stock1 stock2

pg End =
    "cast off"

new pattern Scarf 50
    needles =
        sp
        us 8
    yarn =
        alpaca
        4
    gauge = (8.0, 6.0)

Body
End
```

B.

```
gr Checkerboard =
    color orange = (255, 165, 0)
    color blue = (0, 0, 255)

    gridChunk checkerboard =
        (orange 1 blue 1)+
        (blue 1 orange 1)+

    checkerboard 10

pg Body =
    row knit = k+
    repeat 50 knit BicolorPattern

pg End =
    "cast off"

new pattern BicolorTube 30
    needles =
        ro
        8.0 mm
    yarn =
        acrylic
        3
        blue, orange
    gauge = (10.0, 10.0)

Checkerboard
Body
End
```

C.

```
pg Ribbing =
    row rib = s1 k1 (p2 k2)+
    repeat 16 rib

pg Body =
    row stock = s1 k+
    repeat 75 stock

pg End =
    \cast off"

new pattern LegWarmer 60
    needles =
        ro
        us 6
    yarn =
        merino
        3
    gauge = (10.0, 8.0)

Ribbing
Body
Ribbing
End
```

0.4 Language Concepts

The user will be thinking in terms of paragraphs consisting of text instructions and, optionally, grids defining more complex sequences of stitches. Each paragraph of text instructions will correspond to a part of the garment. Furthermore, each paragraph of text is made up of instructions, and these instructions are themselves made of up sequences including stitches, numbers, or other more specialized directives. For example, one paragraph of a sock knitting pattern might look like this.

```
Cuff:
cast on 64

Row 1: knit 2 purl 2 to end of row.
Repeat Row 1 for 16 total rows.
```

The primitives of this language are integers, floats, and strings. These primitives will all combine in various ways to create the paragraphs of text instructions in the final pattern. The floats will be used to construct colors, which will define rows of colors that can be built up to create the grids (if any) that appear in the final pattern. When a user is programming the paragraphs of text instructions, they will need to ensure that the numbers in the instructions they provide match the number of stitches they've instructed the knitter to use at that point in the knitting process. For instance, if the user attempts to generate the following instruction:

```
Cuff:
cast on 64

Row 1: knit 2 purl 1 to end of row.
Repeat Row 1 for 16 total rows.
```

The parser would recognize that 64 is not divisible by 3 and throw an error.

0.5 Syntax

Each program begins with the statement "new pattern" followed by a string, and then an integer. This string defines the name of the knitting pattern, and the integer defines the number of stitches to be casted on for this pattern. Underneath the "new pattern" statement, the user must define three fields – needles, yarn, and gauge, in that order – by typing the name of the field followed by '='. The needles field defines the type of knitting needles to be used and their sizes, the yarn field defines the type of yarn to be used and its size, and the gauge defines the number of stitches per inch of fabric. All of the above constitutes the header of the program.

Once the header has been completed, the user will leave one line of whitespace and begin typing the body of the program. The body consists of paragraphs and grids. Most of the time, it will be convenient to create a variable for a paragraph or grid above the header.

A paragraph is defined by "pg" followed by a string, followed by a '='. This string is the name of the paragraph. Underneath the "pg" statement, the user will define a string, a row, a repeat statement, a taper statement, or some combination of these. A row is defined by "row" followed by a string, followed by a '=', followed by a sequence of stitches. A repeat is defined by "repeat" followed by an integer, followed by one or more rows. Here, the integer defines the number of times to repeat the rows in the statement. A taper is defined by "taper", followed by a char, followed by an integer, followed by a -¿, followed by another integer, followed by one or more rows. Here, the integers around the arrow define the starting and ending 'widths' of the taper pattern; this instruction is useful for creating rounded portions of garments (like the top of a beanie).

A grid is defined by "gr" followed by a string, followed by a '='. This string is the name of the grid. Underneath the "gr" statement, the user will define one or more grid chunks and the number of times to repeat them in the overall grid. These grid chunks are defined by "gridChunk" followed by a string, followed by a '=', followed by one or more grid rows. Grid rows are defined as one or more color sequences between parentheses, optionally followed by a '+'. Color sequences are a color followed by an integer. A color is defined by "color" followed by a string, followed by a '=', followed by three comma-separated integers between parentheses.

Minimal Formal Grammar
For now, the start expression is stitchSeq.

```
<stitchSeq> ::= <stitch><number>
              |   <plus><stitchSeq>
              |   <oParen><stitchSeq><stitchSeq><cParen>
<stitch>    ::= k
              |   p
<number>    ::= <d><number>
              |   <d>
<d>         ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<plus>      ::= +
<oParen>    ::= (
<cParen>    ::= )
```

0.6 Semantics

i. The primitive values in KnitPick#++ are integers, floats, and strings.

ii. There are many possibilities, but the main way that values are combined is to build up paragraphs of text for the knitting pattern document. Generally, a user will use one or more stitches (each defined as a name, abbreviation, and weight) that are hard-coded into the language or define their own stitches to build up instructions. These instructions combine into paragraphs that describe how to knit the different parts of a given garment.

Another way that values are combined is to build up grids. This process begins with a user defining one or more colors, which are tuples of three integers. Then, the user will define a row of colors by specifying a color and the number of

times to repeat it in the row. After this, a user can define a chunk made up of many rows – basically, one "image" or portion of the overall pattern. Finally, these chunks are repeated an arbitrary number of times to create the final grid. The grid will be represented by an svg file that is imported into LaTeX to become part of the final pdf output.

Minimal Semantics

Syntax	Abstract Syntax	Type Syntax	Prec./assoc.	Meaning
n	`Number of int`	`int`	n/a	*n* is a primitive. We represent integers using the 32-bit F# integer data type (int32).
'k' or 'p'	`Stitch of char`	`Stitch`	n/a	'k' or 'p' represents either a knit or a purl stitch. We construct a Stitch object by keeping a (small) dictionary of stitches and their abbreviations.
sn	`StitchSeq of char * int`	`StitchSeq`	n/a	*sn* is a combining form that communicates a particular kind of stitch (knit or purl) and the number of times it should be performed.
+*S*	`StitchSeq of char * StitchSeq`	`StitchSeq`	n/a	+*S* is a kind of StitchSeq that indicates that the StitchSeq *S* should be repeated until the end of the row.
(*SS*)	`StitchSeq of StitchSeq * StitchSeq`	`StitchSeq`	n/a	(*SS*) is a combining form that allows StitchSeqs to be chained for more complexity.

iii.

Primitives

```
Type integer = integer

Type float = float

Type string = string
```

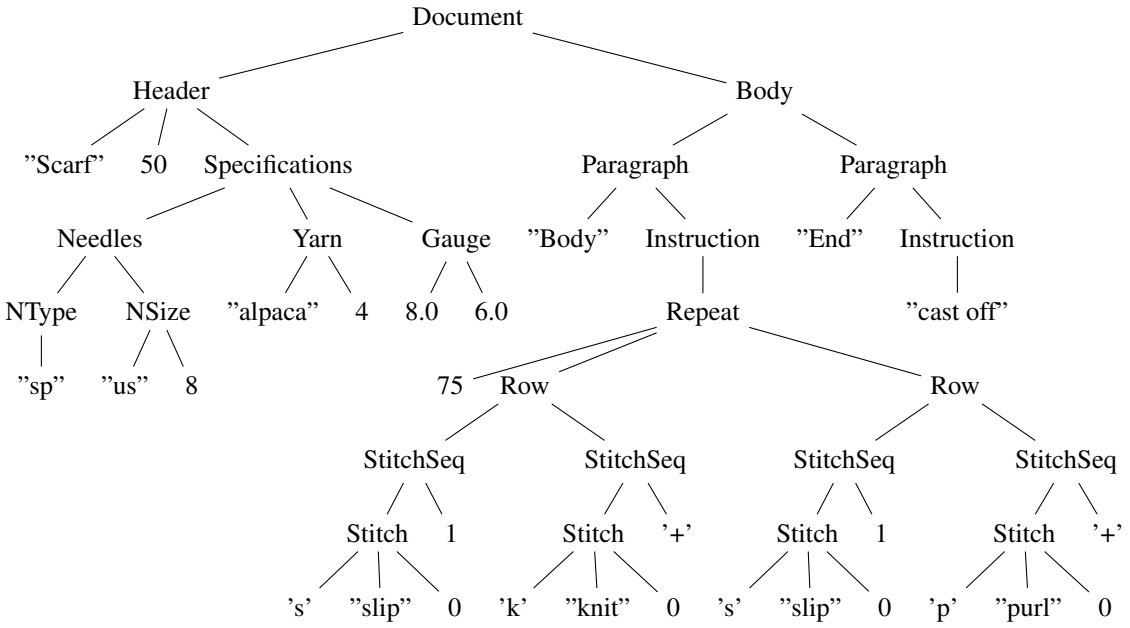
Combining Forms

```
Type Stitch =  string * string * integer

Type StitchSeq =
| Stitch * integer
| StitchSeq * StitchSeq
| StitchSeq * '+'

Type Row = StitchSeq list

Type Repeat = integer * Row list

Type Taper = integer * integer * Row list

Type Instruction =
| string
| Row
| Repeat
| Taper
| Instruction * string

Type Paragraph = string * Instruction list

Type Color = r : integer * g : integer * b : integer

Type ColorSeq = Color * integer

Type GridRow =
    | ColorSeq list
    | ColorSeq list * '+'

Type GridChunk = GridRow list

Type Grid = string * (GridChunk * integer) list

Type NeedleType =
    | \sp"
    | \dp"
    | \ro"

Type NeedleSize =
    | \us" * integer
    | float * \mm"

Type Needles = NeedleType * NeedleSize
```

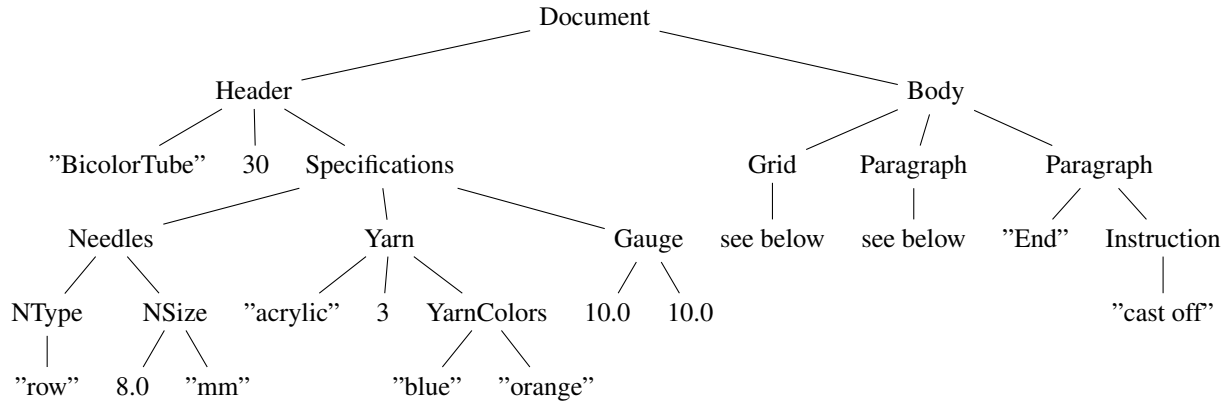
```
Type YarnColors = Color list

Type Yarn =
| string * integer
| string * integer * YarnColors

Type Gauge = float * float

Type Specifications = Needles * Yarn * Gauge

Type Header = string * integer * Specifications

Type Body =
    | Body * Body
    | Paragraph
    | Grid

Type Document = Header * Body
```

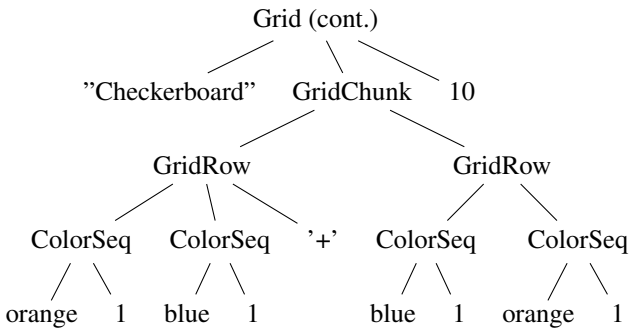
iv. Note: two of the following trees (B and C) were too wide to fit onto the page. In order to represent them, some portions of these trees, below the nodes marked "see below", have been split off and appear below the main tree, marked as "cont."

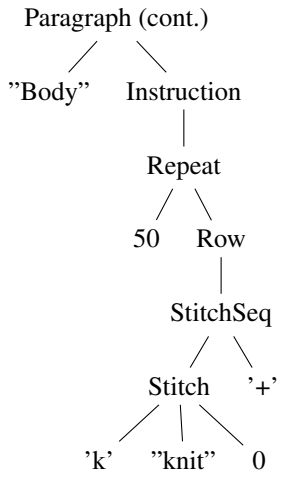
A.



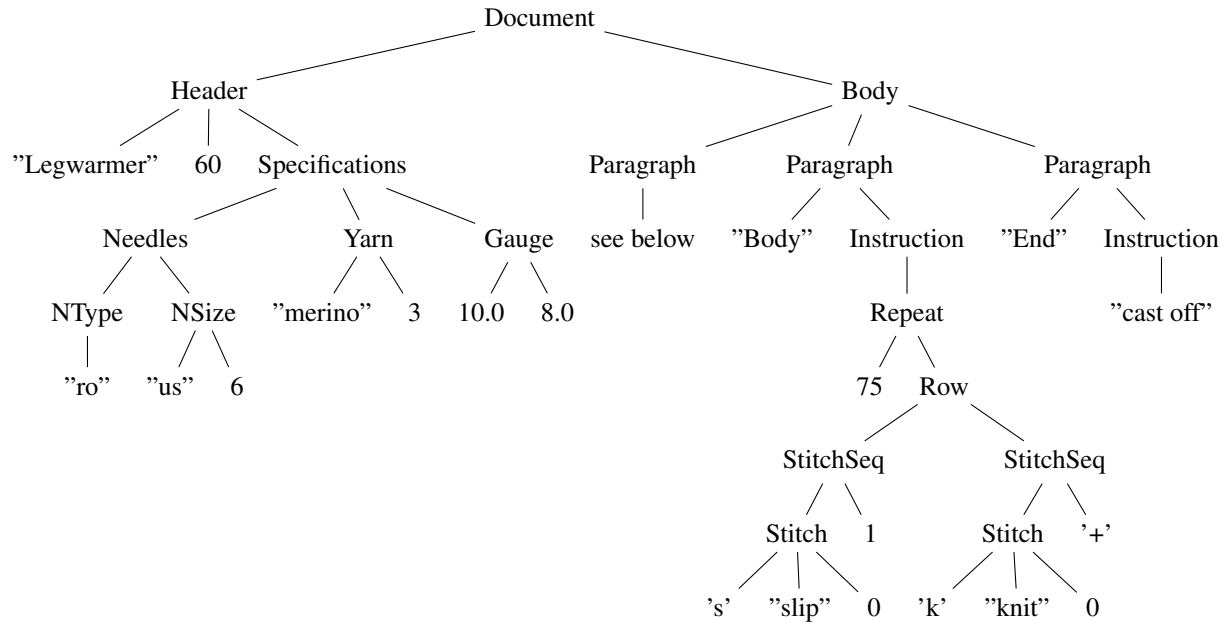
B.

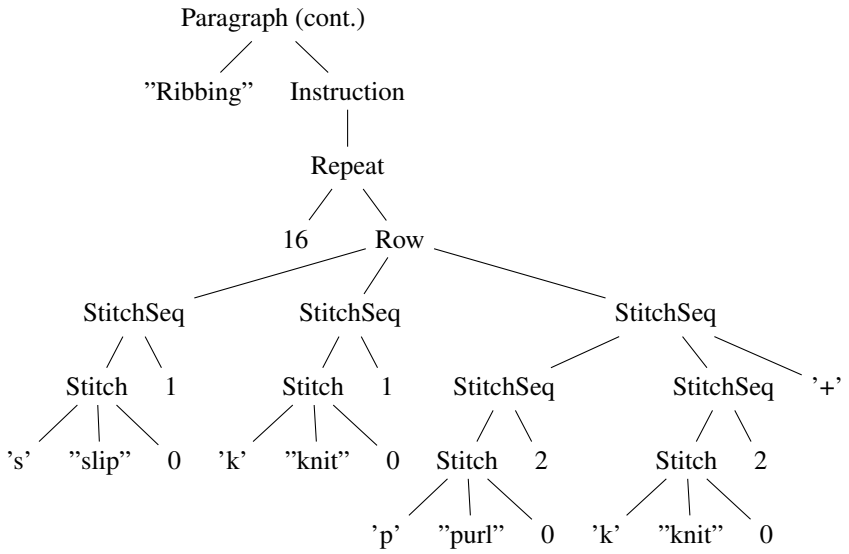

Document
Header
Body
"BicolorTube"
30
Specifications
Grid
Paragraph
Paragraph
Needles
Yarn
Gauge
see below
see below
"End"
Instruction
NType
NSize
"acrylic"
3
YarnColors
10.0
10.0
"cast off"
"row"
8.0
"mm"
"blue"
"orange"


Grid (cont.)
"Checkerboard"
GridChunk
10
GridRow
GridRow
ColorSeq
ColorSeq
'+'
ColorSeq
ColorSeq
orange
1
blue
1
blue
1
orange
1


Paragraph (cont.)
"Body"
Instruction
Repeat
50
Row
StitchSeq
Stitch
'+'
'k'
"knit"
0

C.


Document
Header
Body
"Legwarmer"
60
Specifications
Paragraph
Paragraph
Paragraph
Needles
Yarn
Gauge
see below
"Body"
Instruction
"End"
Instruction
NType
NSize
"merino"
3
10.0
8.0
Repeat
"cast off"
"ro"
"us"
6
75
Row
StitchSeq
StitchSeq
Stitch
1
Stitch
'+'
's'
"slip"
0
'k'
"knit"
0


Paragraph (cont.)
"Ribbing"
Instruction
Repeat
16
Row
StitchSeq
StitchSeq
StitchSeq
Stitch
1
Stitch
1
StitchSeq
StitchSeq
'+'
's'
"slip"
0
'k'
"knit"
0
Stitch
2
Stitch
2
'p'
"purl"
0
'k'
"knit"
0

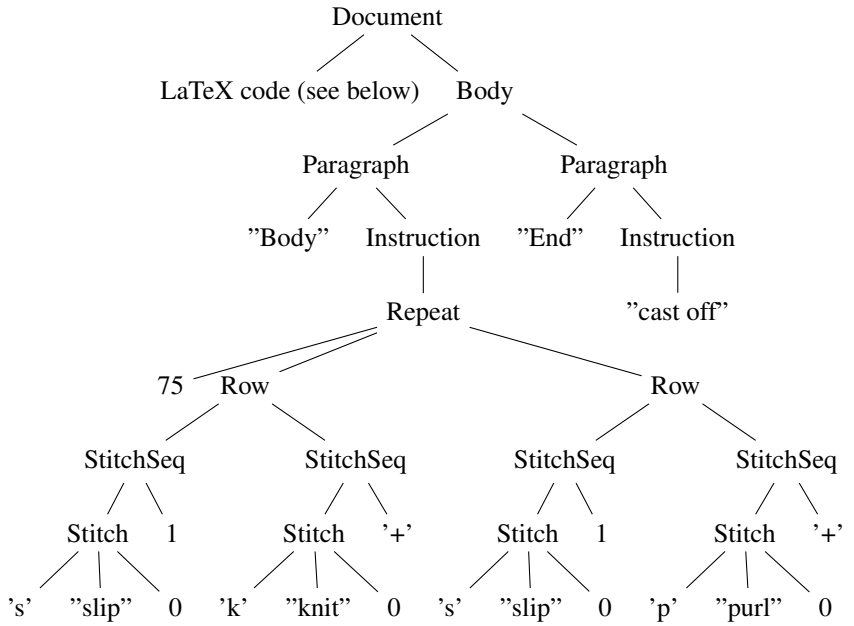
v.

A. Programs in KnitPick#++ do not read any input.

B. The output of KnitPick#++ is a pdf document containing a knitting pattern. This result will be achieved by generating a LaTeX document (with optional svg output) and compiling it.

C. At the beginning of evaluation, KnitPick#++ will set up a simple LaTeX document based on the Header specified by the programmer. This includes several settings that the programmer will not choose, like page style, text size, etc. Since every pattern requires "casting on," that is, creating the first stitches, a "Beginning" section will be generated with the simple instruction to cast on the number of stitches specified in the Header.

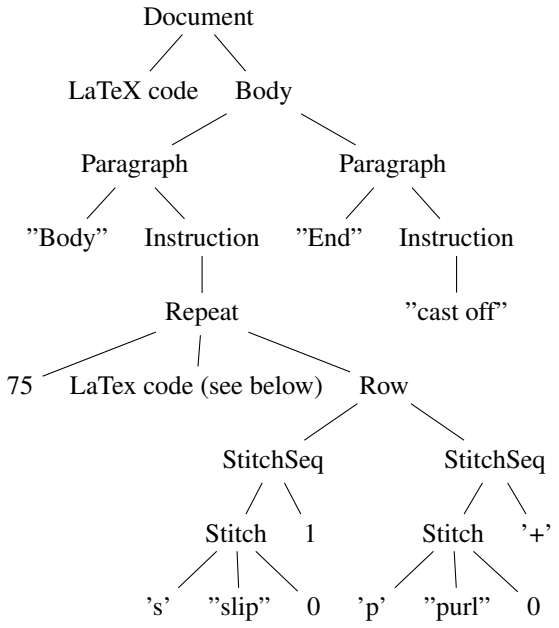
Below is a tree representing the point in the evaluation of example AST (A.) where only the Header branch of the AST has been evaluated and the LaTeX code KnitPick#++ would generate.




```
\documentclass[10pt]{article}

\usepackage{times,graphicx,fancyhdr,amsfonts,xspace,hyperref}
\usepackage[left=1in,top=1in,right=1in,bottom=1in]{geometry}
\usepackage{sect sty} %For centering section headings
\usepackage{enumerate} %Allows more labeling options for enumerate environments
\usepackage[space]{grffile}

\graphicspath{.}

\pagestyle{fancy}

\renewcommand{\headrulewidth}{0.4pt}
\renewcommand{\headwidth}{\textwidth}
\renewcommand{\footrulewidth}{0.4pt}

\setlength{\parindent}{0cm}

\title{Scarf}
\date{}
```

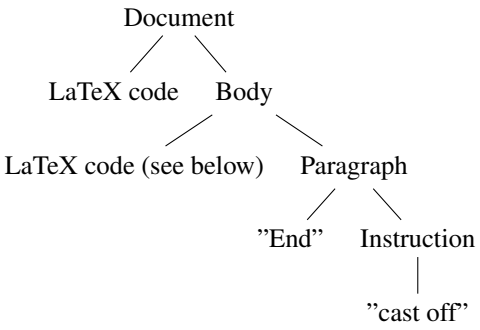
```
\begin{document}

\maketitle

\section*{Project Specifications}

\textbf{Needles}: single pointed size US 8

\textbf{Yarn}: medium size alpaca

\textbf{Gauge}: 8.0 stitches and 6.0 rows per square inch

\section*{Beginning}

Cast on 50 stitches.

\end{document}
```

Next, the rest of the tree would be evaluated to put together paragraphs containing instructions pertaining to the rest of the knitting project. Stitch sequences will be put together into short strings. For example, the bottom left stitch sequence will be made into the following string: ”slip 1 stitch”. The ”+” in a stitch sequence indicates that the knitter should repeat this stitch sequence until the end of the row, so the next stitch sequence (following post-order traversal) will be turned into ”knit to end of row.” These last two stitch sequences will be combined to describe a single row of knitting. Below is the next tree and just the code generated to represent the row just discussed.




```
\textbf{Row 1}: slip 1 stitch, knit to end of row.
```

The same will be done for the other row in the ”Body” paragraph of the pattern, and these two rows will be referred to in the ”repeat” instruction. Evaluating the repeat will result in another string: ”Repeat rows 1 and 2 for 75 total rows.” These two rows and the repeat instruction will be displayed in the Body paragraph using the following LaTeX:




```
\section*{Body}

\textbf{Row 1}: slip 1 stitch, knit to end of row.

\textbf{Row 2}: slip 1 stitch, purl to end of row.

Repeat rows 1 and 2 for 75 total rows.
```

The same will be done for the ”End” paragraph, which only contains one brief instruction. The LaTeX generated for these two paragraphs can be put together in sequence. Finally, the Header and the Body of the document can be combined by placing the code for the Body between just before the end{document} line in the LaTeX document. The full concept LaTeX document (scarf.tex) is in this directory, as is the pdf generated from it (scarf.pdf).