KnitPick#++ Specification

Locke Meyer, Felix King

0.1 Introduction

KnitPick#++ turns the tedious process of creating a knitting pattern, a set of instructions that 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 organise the document clearly; 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 define new stitch types (and other variables) 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. An aesthetic goal is for KnitPick#++ code to be concise and readable; for this reason, many keywords in our syntax are abbreviations.

0.3 Examples

Each of the following examples will be executed by running the command-line argument ”dotnet run example-1.kpp example-1”, and then ”pdflatex example-1.tex”. Sample outputs can be found in the ”examples” folder in the docs directory.

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. Each paragraph of text will correspond to a part of the garment – for instance, the ribbing of a scarf or the cuff of a sweater. Furthermore, each paragraph of text is made up of **instructions**, and these instructions are themselves made up of **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. Integers are used to represent the number of repetitions of a stitch and the number of times to repeat an instruction, among other things. Strings represent the names of stitches and any custom instruction the user wants to include. Floats appear in the header of a document and provide precise information about the ”gauge” of the pattern and the size of the yarn to be used.

0.5 Formal 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).

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

```
<paragraph>   ::= "pg "<string>((<ws>)*<instruction>(<ws>)*)+
<instruction> ::= <string>
                |   <row>
<row>         ::= (<stitchSeq>(<ws>)*)+
<stitchSeq>   ::= <stitch><number>
                |   <plus><stitchSeq>
                |   <oParen><stitchSeq><stitchSeq><cParen>
<stitch>      ::= 'k'
                |   'p'
<number>      ::= <d><number>
                |   <d>
<d>           ::= '0' | '1' | ... | '8' | '9'
<plus>        ::= '+'
<oParen>      ::= '('
<cParen>      ::= ')'
<string>      ::= <quote>(<character)+<quote>
<ws>          ::= ' ' | '\n' | '\t' | "\rn"
<quote>       ::= '"'
<character>   ::= ' ' | '!' | ... | '}' | '~'
```

0.6 Semantics

Syntax	Abstract Syntax	Type Syntax	Prec./assoc.	Meaning
n	`Int of int`	`int`	n/a	*n* is a primitive. We represent integers using the 32-bit F# integer data type (int32).
"*str*"	`String of string`	`string`	n/a	*str* is a primitive. We represent strings using the built-in F# string type.
'k' or 'p'	`Stitch of String * Int`	`string * int`	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	`StitchRep of Stitch * Int`	`(string * int) * int`	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*	`StitchSeqToEnd of 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*)	`TwoStitchSeq of StitchSeq * StitchSeq`	`StitchSeq * StitchSeq`	n/a	(*SS*) is a combining form that allows StitchSeqs to be chained for more complexity.
R	`Row of StitchSeq list`	`StitchSeq list`	n/a	A row combines multiple stitch sequences that, together, make up one row of knitting.
I	`Instruction`	`Instruction`	n/a	*I* is an instruction, either a row of knitting or other direction (for example, "cast on.")
"pg "*name inst*	`Paragraph of String * Instruction list`	`string * Instruction list`	n/a	The string "pg " followed by a string and a list of instructions defines a paragraph. The string is the name of the paragraph and the list of instructions describes how that part of the garment should be knit.
"needle " *ty sys size*	`Needle of String * String * Int`	`string * string * int`	n/a	The string "needle" followed by two strings and an integer represents the needles needed for the knitting pattern. *ty* represents the type of needle, *sys* represents the measurement system (either US or mm), and *size* is the size in that system.

Syntax	Abstract Syntax	Type Syntax	Prec./assoc.	Meaning
”gauge ” (*stitches, rows*)	`Gauge of Float * Float`	`float * float`	n/a	The string ”gauge ” followed by a tuple of floats in parentheses communicates the ratios of stitches to horizontal inches and rows to vertical inches that the pattern designer used.
”yarn ” *”type” size*	`Yarn of String * int`	`string * int`	n/a	”yarn ” followed by a string and and integer represents the type / brand and the size of the yarn used in the pattern

0.7 Remaining Work

We are still planning on implementing a couple of other possibilities for the Instruction type – Taper and Repeat. These cases will allow for easy inclusion of common knitting instructions describing how to create a taper (e.g. the top of a hat) or several repeated rows. After this, we will create the Document type, which is made up of a Header and multiple Paragraphs; we will also need to implement the Header type, which contains information about the types and sizes of yarn to be used for the garment. If we have time remaining after doing this work, we will implement variables so that it’s easy to repeat Paragraphs in a Document.