Error in the Open Source code for Prusa Slicer

I discovered an error in the Open Source code for the Prusa Slicer that causes errors with the Feedrate value and can also cause a related assert statement to fail.

I've been working on a project that modifies and adds new features to the Prusa Slicer. In particular it generates additional GCode to control the printer when the GCode file is used as input to the printer.

One of the features of the GCode protocols is that it is perfectly valid to include comments in the GCode file. You do that by preceding any comment with a semicolon, ';'. You can have a comment on a line by itself and you can have a comment on the same line as a GCode command by just adding a semicolon after the full GCode command and then adding the comment. You might, for example, do something like:

G0 E-0.400000; RCG Filament Retract

In that case it's a filament retraction command that I have added to the Prusa Slicer as part of the new functionality I'm adding to the Prusa Slicer. I add comments like this to help identify the new GCode in the resultant GCode file in order to assist me in debugging any potential problems during development. The "RCG" is my initials and the "Filament Retract" just helps me, or anyone else assisting in the development, to easily understand what the GCode does.

THE PROBLEM . . .

with the existing Prusa Slicer Open Source code is that it fails to recognize comments that are on the same line as a GCode command. It does properly ignore/bypass comments that are on a line by themselves, but not comments on the same line, like the one above. And by "fails to recognize comments" I mean that it doesn't recognize the existence of the semicolon and instead reads all of the words in the comment and attempts to interpret them as parameters to the GCode command.

THE BAD OPEN SOURCE CODE

The source code that does this incorrect processing is in the CoolingBuffer.spp file, within the :

std::vector<PerExtruderAdjustments> CoolingBuffer::parse_layer_gcode(const std::string &gcode, std::array<float, 5> &current_pos) const

method, at about line 428, where it's doing this:

```
// Parse the axis.
size_t axis = (*c >= 'X' && *c <= 'Z') ? (*c - 'X') :
        (*c == extrusion_axis) ? AxisIdx::E : (*c == 'F') ? AxisIdx::F :
        (*c >= 'I' && *c <= 'K') ? int(AxisIdx::I) + (*c - 'I') :
        (*c == 'R') ? AxisIdx::R : size_t(-1);
if (axis != size_t(-1)) {
   //auto [pend, ec] =
```

```
            fast_float::from_chars(&*(++ c), sline.data() + sline.size(), new_pos[axis]);
        if (axis == AxisIdx::F) {
            // Convert mm/min to mm/sec.
            new_pos[AxisIdx::F] /= 60.f;
            if ((line.type & CoolingLine::TYPE_G92) == 0)
                // This is G0 or G1 line and it sets the feedrate. This mark is used for reducing the
duplicate F calls.
                line.type |= CoolingLine::TYPE_HAS_F;
        } else if (axis >= AxisIdx::I && axis <= AxisIdx::J)
            line.type |= CoolingLine::TYPE_G2G3_IJ;
        else if (axis == AxisIdx::R)
            line.type |= CoolingLine::TYPE_G2G3_R;
    }
    // Skip this word.
    for (; c != sline.end() && *c != ' ' && *c != '\t'; ++ c);
```

The variable 'c' is a pointer into the GCode that's being processed, and it picks up and processes all of the words in the current line of GCode that it thinks might indicate what axis the GCode refers to.

FIRST PROBLEM: ONLY CHECKS FIRST CHARACTERS

Actually, it only looks at the very FIRST CHARACTER of such subsequent words in the line of GCode. You could specify "Zanadu" istead of "Z" and it would still process it as if it were referring to the Z-axis.

MAJOR PROBLEM : IGNORES THE SEMICOLON

But the major problem is that it ignores the semicolon. It does not recognize that part of the line of GCode is actually a comment and SHOULD BE TOTALLY IGNORED. Instead, it goes completely through the text of the comment and attempts to interpret each individual word within the comment as if they were valid GCode parameters.

In my example, from above, . . .

G0 E-0.400000; RCG Filament Retract

it interprets both the "RCG" and the "Retract" words as "R" parameters, interpreting them to mean that the GCode is referencing the AxisIdx::R, via the code that does:

*(*c == 'R') ? AxisIdx::R*

and as a result subsequently sets the Line.type incorrectly with . . .

```
else if (axis == AxisIdx::R)
  line.type |= CoolingLine::TYPE_G2G3_R;
```

which it most definitely SHOULD NOT be doing.

More importantly for my particular project, and which caused some very odd behavior and eventually resulted in an assert failure, it interpreted the word "Filament" as being an 'F' parameter, meaning the Feedrate axis, via the code:

*(*c == 'F') ? AxisIdx::F*

THE WORST PROBLEM

But the worst problem (for my particular example) is that, having incorrectly interpreted the comment word, "Filament", as an 'F' parameter, it subsequently does the following:

```
if (axis == AxisIdx::F) {
  // Convert mm/min to mm/sec.
  new_pos[AxisIdx::F] /= 60.f;
```

It takes whatever is the current feedrate, as specified within the new_pos[AxisIdx::F] variable, and divides it by 60!!

Normally this would be the correct thing to do, but ONLY for GCodes that validly, explicitly, include an 'F' parameter. It can be horrible when it is interpreting a word within a comment as an 'F' parameter.

THE EFFECTS

Note that this section of code is within a method that is parsing a potentially huge set of GCode commands (of slines), not just one. In my case it is processing numerous GCode lines that include comments at the end of the GCode command that happen to include the word "Filament" which it is incorrectly interpreting to mean a Feedrate parameter.

The FIRST TIME that it incorrectly encounters the word "Filament" and incorrectly interprets it to mean a Feedrate parameter, the effect isn't so bad, though it still shouldn't do that. It does the :

// Convert mm/min to mm/sec.

new_pos[AxisIdx::F] /= 60.f;

which does indeed calculate a meaningful Feedrate value.

BUT that value contained within new_pos[AxisIdx::F] subsequently AGAIN gets divided by 60 each time the "Filament" word is encountered in a subsequent GCode line (sline) and misinterpreted as a Feedrate parameter. With a sequence of GCode lines (each validly containing a comment section with a word

that's misinterpreted as a Feedrate parameter) that new_pos[AxisIdx::F] get divided over and over again by 60, eventually getting reducd to ZERO and causing an assert failure around line 476:

```
line.feedrate = new_pos[AxisIdx::F];
assert((line.type & CoolingLine::TYPE_ADJUSTABLE) == 0 || line.feedrate > 0.f);
if (line.length > 0) {
   assert(line.feedrate > 0);
```

DEBUGGING

When I encountered the assertion failure I did a lot of debugging, sending diagnostic information out with the std::cout instruction. In particular I sent out the value of new_pos[AxisIdx::F] each time it was calculated by the

```
// Convert mm/min to mm/sec.
new_pos[AxisIdx::F] /= 60.f;
```

For the portions of GCode that are generated by the normal processing of the Prusa Slicer everything went fine. But when it encountered the additional, new GCode commands that I have added in my modification to the Prusa Slicer code, I could see that the new_pos[AxisIdx::F] value simply got smaller and smaller and smaller until it hit zero, resulting in the assert failure.

And it most definitely is the result of this particular method:

std::vector<PerExtruderAdjustments> CoolingBuffer::parse_layer_gcode(const std::string &gcode, std::array<float, 5> &current_pos) const

in the CoolingBuffer.cpp

that FAILS to recognize comments on valid GCode command line, instead interpreting every single word in the comment as a potential parameter to the GCode command.

Now, THERE IS a section of code that detects the presence of a semicolon in an sline:

```
// Skip whitespaces.
for (; c != sline.end() && (*c == ' ' || *c == '\t'); ++ c);
if (c == sline.end() || *c == ';')
   break;
```

HOWEVER, that only occurs at the very beginning of the processing of each sline (i.e. current line). It validly skips over blank lines and lines that BEGIN with a semicolon (indicating a line that is JUST a comment.

What is missing is detection of semicolon in lines that start with a valid GCode command but also include a comment on the same line AFTER the GCode command.

WHAT SHOULD BE DONE . . .

1) AT THE VERY LEAST it should do more than look at the first character of each word and try to interpret it as a parameter. "Filament" is not the same as 'F'.

2) It should also detect the situation where (it thinks) it has encountered a parameter specification BUT that specification is NOT followed by a valid numeric value. In my example:

G0 E-0.400000; RCG Filament Retract

The misinterpreted word "Filament" is NOT followed by any numerical value. That should be flagged as an error, possibly detected by an assert. And of course that can occur even for validly specified parameters. A Gcode like:

G0 E-0.400000 F;

is just as incorrect.

3) But the most important issue here is that it is not recognizing (and properly ignoring) comments that are included at the end of a valid GCode command.

Now, an argument can be made that the engineers who wrote the Prusa Slicer are just very careful to never include comments on the same line as a valid GCode command. They, instead, whenever they deem it helpful to have comments included in the generated GCode file, include those comments on individual lines all by themselves.

BUT, all of their source code for the Prusa Slicer is available on GitHub as Open Source source code, meaning that other engineers, potentially lots of other engineers (not familiar with Prusa's approach to this software), can get the source code and attempt to make modifications and additions to it. And those engineers, following the protocols for GCode can perfectly validly create lines of GCode that include comments following the individual GCode commands. And that can result in serious problems, including assert failures.

That's what I did. And it caused much frustration and significant lost development time when it started encountering assert failures, all because of one existing method that doesn't properly check for comments that are perfectly valid according to GCode protocols.