

# 翻訳者は3か国語に対峙する(前編)

岩沢 節

ソフトウェアのマニュアルを英語から日本語に翻訳する仕事に携わる者はしばしば、「この言葉は英語なのか、プログラミング言語なのか」という疑問に直面する。

その原因の根底にあるのは、すでにバイリンガル（英語とプログラミング言語）である著者が、同様にバイリンガルであるソフトウェア技術者に向けて多くのマニュアルを書いていることである。ソフトウェア技術者は「includeする」とか「UNIONをとる」といった概念を互いに分かりきった事柄として、説明なしでどんどん使う。include や UNION（共用体、論理和、和集合）といった語は英語ではあるが、プログラミングの世界で独特の意味を持つ。AND や OR でさえ、独特の意味を持っており、英文マニュアルには ANDing や ORing といった“現在分詞”が登場することさえある。つまり、著者も読者も「これは英語かプログラミング言語か」など、いちいち気にしなくても大丈夫なのだ。

では、ほとんどのプログラミング言語が英語をベースにしているのであれば、「日本人でもプログラミングに通じた技術者なら英文マニュアルなど苦もなく読めるだろう」と思われるかもしれない。しかし、多くの場合、そう簡単にはいかない。日本人読者は英語がそれほど苦手でなくても、忙しくて英文マニュアルなど読む気がしない。さらに、英文原典の著者が日本語や他の言語のことをまったく知らないことも、問題を複雑にしているように思われる。つまり、

英語圏のソフトウェア技術者と日本の技術者は、プログラミング言語という“共通語”だけでは十分に意思を伝わせられないのだ。

そこでマニュアル翻訳者が必要とされるわけだが、ここで問題となるのは、翻訳者がたとえ英語と日本語の完全なバイリンガルであっても、十分ではないということである。理想は、翻訳者がプログラミング言語の理解も持っていてトリリンガルになっていることだが、現実にはプログラマ出身ではない翻訳者も多い。（以前プログラマだった翻訳者にとっても、最近のプログラミング言語特有の概念に習熟するのは大変なことである。）

たとえば、テーブルの定義を include する場合、テーブルの定義全体をそこに長々と打ち込むのではなく、テーブルの定義は別のファイルに記述されていて、そのファイル名を #include 文で呼び出すだけである。そうした基本的なプログラミング作法を知らずに翻訳者が「テーブルの定義を最初に含めます」と漫然と訳すなら読者は戸惑うだろう。

ある程度プログラミングに通じた翻訳者であっても、「ここでの print は普通の印刷の意味だろうか、それともファイルへの出力もありうるとして『プリントする』くらいが無難だろうか」といちいち悩むことになる。そういう場合の一般的な逃げとして音訳が便利であるため、ときに、プログラミング的な意味合いのない言葉さえカタカナにしてしまい、「取り出

す」でよいのに「リトリブする」といった、意味不明の日本語を頻出させてしまう可能性がある。

## long fields ? integer fields ?

longvarchar や int32 といった独特の名前が付いていれば、それはすでに英語でないことは明らかだが、普通の英語の単語が変数やフィールドや列のタイプ名になっている場合もある。さらに、int32 や int64 などのフィールドをひっくめて integer fields と呼ぶこともあり、その場合は総称として「整数フィールド」（あるいは、整数系のフィールド）と訳す必要がある。同様に、long 自体がタイプ名である可能性がある一方、各種の長いフィールドをまとめて long fields と呼んでいる可能性もある。

こうした疑問は、翻訳されたマニュアルの読者だけでなく、英文原典の読者も抱く疑問と思われるのだが、英文読者のプログラマは文脈に沿って適宜解釈しているのに対し、日本語のマニュアルでは「integer フィールド」と「整数フィールド」のどちらか一方で表記されることになる（しかも複数形が訳されない）ので、プログラマが文脈に沿って適宜解釈することは困難になる。翻訳者に重大な責任が負わされているのである。

この点であいまいさを避ける最良の方法は、本文中にキーワード（つまり英語ではなくプログラミング言語）が登場する箇所では、著者が最初から書体を分けることである。「この語はプログラミング言語に属するもので、英語ではない」ということを、モノスペース書体などで明示すること、それが、マニュアルを書く人が常に心がけるべき親切であろう。

（後編に続く）