私はまずリストというものについてほぼ何も知らなかったため、この問に回答するために、実際のコード例と共にリストがどういったもので、どのように実装すればよいのかを勉強することから始まった。

その過程で、リストについて以下の知識を得た。

・双方向リストはポインタで前後の要素のアドレスを記憶しているため前後を自由に行き来できる

・配列のように添字で管理できないのでランダムアクセスはできない

・リストは挿入や削除、追加が配列と比較して簡単にできる

という知識を得たので、早速プログラムを作成していこうと考えた。これらをすべて頭の中で考えるのは難しいと悟ったため、紙に図を描いて大まかなアルゴリズムを考えた。大まかに考えたところでプログラムで実装した。

そこでプログラムを作成する上で私は、大学の授業でアサーションについて習ったため、このプログラムでも活用した。入力値を監視し、不正な値が入力された場合にプログラムを強制終了させるものだ。不正値が入力され、プログラムが変な挙動を起こさないように対策した。また、リストの解放を行う関数を最後に実行するようにした。これは、mallocで動的に割り当てたメモリ領域を開放してやらないと他のプログラムがそのメモリ番地を利用することができなくなってしまうからである。

今回のプログラムは、最初実装したとき、リストに要素を挿入する関数のみ設計したとおりに動かなかった。具体的には挿入したデータとそのひとつ前のデータに変な値が格納されてしまった。(挿入されたデータには巨大な数字,そのひとつ前には0) 原因はInsList関数の最後でnewとnowの構造体ポインタ変数を開放していたからだった。malloc関数で動的にメモリを割り当てた場合、最後にメモリ解放をしなければいけないと考えていたため、なかなか間違えている箇所に気づかなかった。

間違えている箇所を探すために、まず私が行ったことは、挿入を行った場合と行わなかった場合で出力された結果がどのように想定の結果と変わるのかを比較した。これは、このプログラムの中で唯一InsList関数に入る場合と入らない場合で比較することが可能なためである。InsList関数に入って、要素の挿入を行った場合、先述のような結果となったが、InsList関数に入らず、挿入を一度も行わずに出力した場合、正常に出力されたため、リストの作成とリストを出力する関数、main関数には不具合はないと考えた。そのため、私の考えた、リストを交換するアルゴリズムとInsList関数のどちらかに不具合の原因が存在すると考え、両方のトレースを行った。

これは、私の考えたアルゴリズムが間違えているのか、それとも実装ができていないのかを明確にするためである。トレースの結果、リストに要素を挿入するアルゴリズムに不具合を見つけることはできなかった。InsListをトレースし、見つけた不具合の修正や、不要なコードをリファクタリングした。関数の処理の順番を変更してみたり、ポインタの切り替え部分をトレースし直してみたりした。出力時に現在のアドレスと、構造体に格納されている次のアドレスと前のアドレスを表示するようにした。その結果、リスト構造は正常に成立していることが確認できた。しかし、データは先述の通り正しく格納されていなかった。Webで「リスト 挿入」で検索し、いくつかのコード例を見ていたところ、関数の最後で解放していないことに気づいた。試しにコメントアウトしてみたところ、正常に動作した。