言語処理系の開発にあたって

954480 岡本悠里
第 1 版　2022/01/30
第 2 班　2022/01/31

この課題を行って会得したことや感想などを三千字(3,000 字)以上

　この授業で学んだプログラミング言語は、パラダイムを学ぶものとなった。プログラミング言語には関数、論理などスタイルが決まっている場合が多いと感じた。マルチパラダイム言語でもどのパラダイムを採用するかを決めている。今回の課題で言語を作るとなった時に、新しいパラダイムを考案しなければ新しく言語を作る意味がないと思い、新しいパラダイムを考え出そうとした。言うまでもなく、苦戦し、パラダイムを作り出すとは何なのかについて考えざるを得なくなった。いくつかのパラダイムを眺めているうちに、パラダイムというのは作ろうとして作れるものではないのではないかと考えた。困ることや要望があってそれを解決する手段として考え出されるという流れを踏み、先にパラダイムを作り終えてその後にどういった要望が叶えられるかという順ではないのはないかと思った。つまり、この課題を取り組むには日頃困っていることなどを洗い出し、それらを解決するような言語を作るという流れが自然ではないかと考えた。よって、今回はパラダイムを考え出すことまでは出来なかったが、英数字の羅列が怖いと思う困りごとを想定し、これを解決するような言語を作ろうということになった。更に、自分の経験から値の代入または束縛において 1 回生の時に感じた「=」を使う違和感について解消しようと考えた。統合できないものに=を使う事に違和感を感じる上に、代入と束縛の違いが全く理解できなかった。視覚的に最初から荷札だと思わせてしまえば良いでのはないかと考えた。つまり、この言語を使うユーザを玄人ではなく素人とした。yacc や lex を用いながら自分がデザインした言語を実装していく体験ができるということはもちろんなのだが、実際に世の中に出た際に誰が使うのだろうかということを考える機会になった。先生の授業を聞いていきなり全部実装するというのはやめて、少しの機能から実装しなさいというアドバイスをもとに最初は定義と足し算引き算しかないものを作った。たかが足し算引き算だが、実装は困難を極めた。この僅かな機能であっても起こりうる想定外の事態がたくさん考えられた。決めた文法に沿って定義が行われるように、その定義を踏まえて正しく計算できるようにするだけでもこの場合はうまくいくべきであり、この場合はエラーにしなければなど不具合の削減が難しい。今まで行ってきたソフトウェア開発でのテストとは違っ

たテストであると感じた。単体テストや結合テストなどだけでなく、そもそもの言語を作っているので、起こりうる入力パターンを確実に掬い上げる必要がある。テストケースをできるだけ多く作り出し、あらゆる脅威に対応することの重要性と難しさを学んだ。私は最初、lex で文字列の定義をクオート(‘)で囲まれたすべての文字を文字列と定義していた。考えられる入力を考えているうちに、クオートが含まれた文字列が渡された時にうまくいかないことに気づいた。文字列でクオートを使いたい時に困ってしまう。たかが定義、たかが文字列、たかが print だがこのように不具合が発生する。結局、¥’と入力しエスケープさせることでクオートも文字列として定義することができるようになった。少ない機能にもかかわらず次々不具合がでてきた。一気に while やら for やら if を注ぎ込んで作成していたら処理系を作ることにも至らず、一度も動くことがなかったのではないかと思う。時間や技術の限界で足し算引き算 print ほどの小さな機能しか織り込めなかったが、プロンプタをターミナルに出し、足し算や引き算を行うことができる言語を作ることができた。多くの言語がバージョンアップを重ねながら機能を増やし、不具合を修正していくように、この言語でも完璧に 1 から作ろうとせずに、少しずつ段階を踏みながら成長させていくという考えが重要であると考える。

言語を作成するにあたって、今回もドキュメントを作ることにした。ソフトウェア工学からの積み重ねによってドキュメントを作る流れがどんどんスムーズに、かつ、うまくまとめられるようになってきていると感じる。初めて作成した時はかなりの時間を費やし、作ることに抵抗もあった。特に html はプログラムの一種であるため、余計抵抗があった。苦しい時を過ごし、未熟ながらも 1 度完成させ、更に長期課題 1 でもドキュメントを作成するなどといった反復により、ドキュメントを作る流れが染み付いてきた。誰がどのように作ってきたのかをきちんと示し、どのような設計図を経て完成まで至ったのかを示すことで、万が一の時に備えることができる。たとえ課題が完成しなくても出来た部分までを示すことができる。誰かから権利を侵害されるような行為を取られたとしてもどちらが先に作ったもので、オリジナルがどちらかを示すことができる。ソースコードをただ提出するだけではなく、開発の取り組み方や変遷を示し、どこまでテストしたのかを示すことは自分達が行ったことを他者に伝えるだけではなく、今後の自分にも役立つのではないかと気づいた。今回ドキュメントを作成するにあたって過去のものを参照した。過去の自分や過去のチームメンバーが何をしていたのかを振り返ることができた。チームメンバーはプロジェクトごとに変動する。今回は人数もかなり減ったため、自分の担当する部分も増えた。今回はドキュメントのほぼすべてを 1 人で書くことになった。あれだけ抵抗を感じていたドキュメントかつ html 形式のものをほぼ 1 人で書き切ることができたことには驚きさえ感じる。これを支えたのは過去のメンバーが書いた遺産である。過去のドキュメントを見ると、今まで自分がやっていなかった仕事を誰がどのようにやっていたのかを知ることができる。これは多いに助けになった。未来の自分や他者の助けになる可能性を持つものがドキュメントであるのではないかと思う。今回の課題を含め、今まで自分とその仲間が作ってきたドキュメント

が誰かの助けになるのかもしれない。誰がどれだけやったのか、どのように取り組んだのかという「課題」という狭い枠組みの中での「評定」を公正につけてもらうというその場だけ有効なものではない。記録し、誰かに伝え、遺すという行為の重要性とそのやり方について体感できた。このドキュメントを作るという行為は他の教科に活き始めている。全く他の教科でも、グループワークやプロジェクトのような長い時間をかけて取り組むようなものでは、なるべく記録を残し、作業過程を”レポート”するようになった。結果主義ではなく、経過まで報告するのがきちんとした報告であるという考えに変わりつつある。そしてその残し方は一連のドキュメント制作過程で身についてきている。リポジトリでバージョン管理を行い、いつどの機能が完成したのかという記録をつけることで、ドキュメントもより書きやすくなる。様々な手段を使いながら、記録に残していくという武器を会得しつつあると感じた。

今回作成した言語は Python を使って仮想マシンを作り、仮想マシンを利用することで実行できるようにした。(‘ 8 ‘*)は 1 人では動けないということになる。yacc や lex を用いている時点で他の言語を使用し、C も使っていることになるが、仮想マシンでは Python に大きく力を借りている。Python は C の力を借りて動いている。自作した言語だけでなく、世界で使われる言語は何かしらの言語の力を借りている。C でさえもアセンブラを使う。最終的には 0 と 1 の ON/OFF の電気信号になるまで頼り続ける。(‘ 8 ‘*)を作っている時に、こんなに Python を頼りにしてしまっていいのだろうかという疑問が湧いた。見た目だけ変えて中身はすべて Python ではないかと。しかし、確かに小さな言語であるため大きく依存しているが、Python であっても C を使うなど他の言語に依存していると気づいた。言語の多くは何かしらの過去の遺産を使い、新しいものを生み出していく。逆に 0 と 1 の ON/OFF の電気信号だけでは活用できない。糖衣に包み、包んで色々な言語が誕生している。授業でプログラミング言語の高い山を急に引きずられるように登った上に、降ろされ、1 から山をつくれというのがこの課題である。山を登る最中の景色や、登り方、山頂からの景色を思い出しながら山の模型程度のものをつくったことになるが、世の中の言語が上記のように依存し合っていることに気づいた。これによって、特定の言語の文法だけを学ぶことの実のなさをより感じることができた。他の言語に依存しながら、パラダイムという考え方が違うものまであるのだから、1 つの言語に固執することは勿体無いことである。しかし、こう感じた結論を他人に理解してもらうことは難しい。やはり、一度踏ん張って山を登らないと納得できない。ましてや 2 回生春までのプログラミング言語の授業で癖がついてしまった人には尚難しい。しょうもない機能しか実現できなかった言語作成が、しょうもないことですら作ることが難しいことを体感させられる。授業で教えを聞いた後に実際に課題で実践し、授業中にこっそり張られていた伏線を回収しながら課題を解いていく。これで共通教育科目を抜くすべての青木先生の授業を受講したことになるが、確かに潜って授業を受けるだけでもいいから受けろとは思うが、やはり、課題までやると完成されるようにできていると感じた。しんどく、難しい課題であるのは事実なのだが、どうにか試行錯誤しているうちに何

かに気づくことができる。それが完成まで行かなかったとしても、完全な理解にならなかったとしても、必ず昨日の自分より強くなっている。毎週大量のレポートを書くうちにレポートの書き方がわかること、プロジェクトをいくつもやるうちに開発の流れがわかること、定期的に来る課題に対してスケジュール管理することなども課題をやらないとわからないことである。無意味なしんどさが無いことがより真剣に取り組み、より時間をかけようとする動機につながっているのだと思う。

正直な話、どう考えてもこの分野は興味がない。言語を定義するなどというのは他の人が頑張ってくれればいい。私はデザインがやりたい。入学前からずっと考えていたことだ。1回生の時に私はすべての科目を見て、履修する科目を決めていた。コース制度があるため、それに沿いながら興味のあるものを取り、計算し、卒業できるようにプランを立てていた。そもそも大学に通えるかという不安もあったので最低限頑張ろうと思っていたのもあるが。応用プログラミングという名のつく物は真っ先に除外された。基礎プログラミング演習で周りが授業中の演習を終えて次の授業の分まで手をつけているのに私はいつまで経っても進まなかった。こんなやつには無理だと思ったので、もちろんプログラミング言語も除外された。ただ、Python だけは知っておきたかったというだけだった。つまり、プログラミング言語という科目は青木先生が担当でなければ一生取ることは無いほど興味がない分野だ(正直)。画面と睨んで 1 日過ごすより山に行って川に行ってキャンプやら釣りやらして、自然とアートを混在させるとかそういうことがやりたい、と思っていた。しかし、この授業は他の授業と連動し続ける。ソフトウェア工学で PM という役職を知った時は普通に前から好きなことであったので、軽い気持ちでそのままこの授業を受けた。内容は普通に興味がないのにあれもこれもどんどん繋がっていく。他の授業がつながっているだけでなく、自分が興味のある分野とも繋がった。細かい部分は他人がやればいいと思っていた部分に触れることができた時、自分が元々興味のあったものに対する視点も変わってきた。今回の課題も今までの授業も含めて学ぶということは何なのかについてかなり前進したと思う。まさか言語を作ることになるとは思わなかったが、言語の作成を通して、言語の作成方法といった各論ではなく、他の景色も見えることとなった。