学生の書いたテストプログラムの Coverage ツールを用いた解析

安土　茉鈴　(指導教員：浅井　健一)

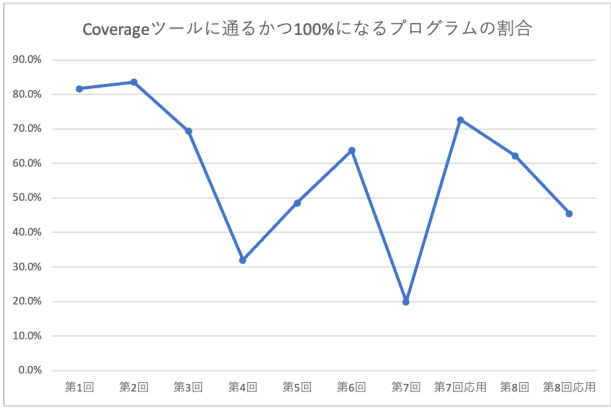
1　はじめに

昨今、コンピュータがウイルスに感染するなどの問題が大きくなっている。その原因としては、プログラムにバグが生じていることが挙げられる。そこで、プログラムの信頼性を上げることが非常に重要になっている。そのために関数型言語 [2] の授業では、デザインレシピ [1] を用いてテストプログラムを書くことの重要性を語っている。本研究では、bisect_ppx という Coverage ツールを用いてプログラムすべてをカバーするテストが書けているかを確認することで、次年度以降の関数型言語の授業でプログラムの信頼性を上げられるようにすることを目的としている。

2　分析手法と全体像

本研究で用いた Coverage ツールとは、実際にプログラムが実行されることで、各々の部分を実際に実行できるようなテストが書けているかを確認できるツールである。手法としてはまず初めに、Coverage ツールに通らないプログラムを除外ファイルとして分類した。その後、残りのプログラムを全て Coverage ツールに通し、どのぐらいの生徒がプログラムの中をきちんと通るかを確認するテストが書けているかのかを求めた。また同時に 100%でなかった場合には、何が原因であるのかも分析した。ただし、同じ人が何度も同じ課題を出している場合もあるので、母数は必ずしも生徒の人数ではない。なお本研究では、2022 年度関数型言語の授業の第 1 回～第 8 回応用課題までを分析対象としているが、これは第 9 回以降の最短路問題を求めるような課題では実行に時間がかかり過ぎてしまって現実的ではないからである。

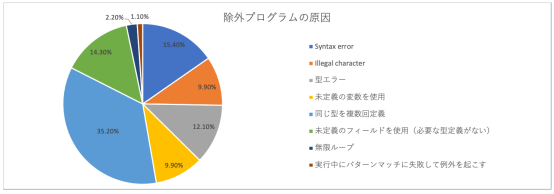
各回ごとに提出された課題のうち、Coverage ツールに通るかつ 100%になるものの割合は下図のようになった。



回を重ねるごとに 100%になるものの割合が低下しているというわけではないが、著しく低い第 4 回と第 7 回は特に注意してみていきたい。

3　各回課題分析の結果と考察

まず初めに、Coverage ツールに通すことのできなかった除外プログラムについて下図にまとめておく。



これらのものについては、まずはコンパイルできるプログラムを提出するよう、授業内で強く呼びかける必要がある。ただし同じ型を複数回定義しているものについては、インタプリタ実行はできるが、Coverage をとるために必要なコンパイルができないので、今回は考察の対象としていない。

3.1　第 1 回課題

第 1 回は関数を定義する課題である。Coverage に通ったもののうち、58/64 が 100%チェックできるプログラムが書けていた。テストなしのものが 4/64、テストの書き方が不十分なものが 1/64、テストは書こうとしていたが、テストで引数の渡し方を間違っていたため型エラーになってしまい、テストが実行されなかったものが 1/64 であった。ここから第 1 回の授業では、授業中にテストを書く指示をしていないにも関わらず、比較的多くの人がテストプログラムを書いていることが分かる。

3.2　第 2 回課題

第 2 回は if 文を使って条件分岐をする課題である。この回からデザインレシピが導入され、テストを書くことが必須になった。Coverage に通ったもののうち、51/57 が 100%チェックできるプログラムが書けていた。補充の練習問題で日付を与えたら星座を求める問題のように、分岐が多い問題であったためテストで全ての場合をカバーできなかったものが 5/57 あったが、それ以外は判別式で解が 1 つになる場合のテストが書けていなかったものが 1/57 あったのみだった。ここから、ほとんどの人が if 文の分岐の全ての場合のテストを書けていることが分かる。

3.3　第 3 回課題

第 3 回はレコードのパターンマッチをする課題である。Coverage に通ったもののうち、50/52 が 100%チェックできるプログラムが書けていた。テストを書こうとはしているが、レコードだけ定義して関数を呼び出していないものが 1/52、関数のテストがないものが 1/52 であった。ここから、レコードのパターンマッチについては概ねできていたと言える。

3.4　第 4 回課題

第 4 回はリストからペアを返す関数 romaji_to_kanji2 とペアとリストから距離を返す関数 get_ekikan_kyori2 を作る課題である。Coverage に通ったもののうち、100%チェックできるプログラムを書けていたのは 24/63 にとどまった。このうち、テストがなかったり、不適切だったものが 5/63 あり、それら以外は全

て if 文の条件を網羅できていないものだった。網羅できていないテストは、空のケースをテストできていないものが 27/63、リストの先頭しか考慮できていないものが 12/63、それ以外は、get_ekikan_kyori2 で複雑な条件をテストできていないものが 22/63 だった。get_ekikan_kyori2 でテストできていなかったケースは、典型的には同じ都道府県の場合のテストしか書けていないものである。if 文で場合分けした際は、各々がどのような場合なのかを考え、それに合わせたテストを書く必要がある。

3.5 第 5 回課題

第 5 回は挿入法と重複を取り除く関数 insert2 と seiretsu2 を作る課題である。第 5 回も Coverage は低かった。Coverage に通ったもののうち、100%チェックできるプログラムを書けていたのは 36/64 にとどまった。このうち、テストがなかったものが 2/64 あり、それら以外は全て if 文の条件を網羅できていないものだった。網羅できていないテストは、空のケースをテストできていないものが 4/64、リストの先頭しか考慮できていないものが 2/64、それ以外は、insert2 で複雑な条件をテストできていないものが 23/64 だった。insert2 でテストできていなかったケースは、典型的には先頭の都道府県よりも残りの都道府県の方が大きいものしか含まれていない場合のテストしか書けていないものである。やはり条件分岐を用いる際には、全ての場合を考えるのが重要である。

3.6 第 6 回課題

第 6 回は条件を満たしたリストを作成する関数 shokika2 を作る課題である。Coverage に通ったもののうち、44/53 が 100%チェックできるプログラムが書けていた。このうち不適切だったものが 3/53 あり、テストを書こうとはしているが、shokika2 で初期化する駅が駅名リストに含まれていなかったものが 1/53、shokika2 で空の場合のテストができていないものが 5/53 であった。この課題は比較的簡単だったので、テストもよくできていた。

3.7 第 7 回課題

第 7 回は更新関数 koushin1 と koushin を作る課題である。Coverage に通ったもののうち、100%チェックできるプログラムを書けていたのは 10/37 にとどまった。このうち、過去の課題でテストがなかったり、不適切だったものが 25/37 あり、それら以外は全て if 文の条件を網羅できていないものだった。網羅できていないテストは、koushin1 や koushin で複雑な条件をテストできていないものが 13/37 だった。koushin1 や koushin でテストできていなかったケースは、典型的には match 文で分けた rest に最短距離が含まれている場合のテストしか書けていないものである。ここから条件分岐をする際には、どんな場合分けをしているのかをメモしながら進めていくことが有効であると考える。

3.8 第 7 回応用課題

第 7 回応用は第 6 回で 2 つの関数で行っていたことを 1 つの関数で行う make_initial_eki_list2 を作る課題と List.map を使う課題である。Coverage に通ったもののうち、32/41 が 100%チェックできるプログラムが書けていた。このうち、過去の課題でテストがなかったり、不適切だったものが 7/41 あり、make_initial_eki_list2 で空の場合のテストができていないものが 1/41、make_initial_eki_list2 で起点が見つかる場合のテストがないものが 2/41 であった。この課題は比較的簡単だったので、テストもよくできていた。

3.9 第 8 回課題

第 8 回は最短距離が最小になるものを分離する関数 saitan_wo_bunri2 を作る課題である。Coverage に通ったもののうち、33/48 が 100%チェックできるプログラムが書けていた。このうち、過去の課題でテストがなかったり、不適切だったものが 10/48、空の場合のテストができていないものが 3/48 だった。それ以外は、リストの中で最小のものが現れる場所 (最初、真ん中、最後など) を十分にテストしきれていないものが 4/48 あったが、全体としては悪くない結果であった。これまでの課題とは違い、条件が複雑になってきたことで、回答が想定されていたものとは異なるものも多くあり、テストが不十分なだけではなく、プログラム自体が間違っているものも見受けられた。

3.10 第 8 回応用課題

第 8 回応用は最短分離を List.fold_right を使って書く関数 saitan_wo_bunri2 を作る課題である。第 8 回と本質的にやっていることは同じであり、Coverage の傾向も似ていた。Coverage に通ったもののうち、15/32 が 100%チェックできるプログラムが書けていた。このうち、過去の課題でテストがなかったり、不適切だったものが 13/32 だった。それ以外は、saitan_wo_bunri2 でリストが昇順または降順に並んだ場合しかテストがなかったものが 3/32 あったが、第 8 回と同様、全体としては悪くない結果であった。

4 まとめと今後の課題

全体を通して、そもそもテストがないもの、条件分岐に対するテストが不十分なもの、空の場合のテストがないものの 3 つが Coverage が 100%にならない大きな要因であった。関数型言語の授業では、デザインレシピを用いて自分の書いたプログラムを確認するテストを書くことを強調しており、ある程度の部分のテストは書けているが、必ずしも全ての場合のテストを書けている生徒ばかりではないということが言える。今後の関数型言語の授業では、(1) 空の場合のテストを書く (2) 条件分岐する際に、その都度どんな場合分けをしているのかをメモし、それを確認できるテストを作る (3) 必ずコンパイルが実行できるかを確認するという 3 つの手順を徹底させていくと良いと思われる。

参考文献

[1] Felleisen, M., Findler, R. B., Flatt, M. and Krishnamurthi, S.: *How to Design Programs: An Introduction to Programming and Computing*, The MIT Press (2018).

[2] 浅井健一：プログラミングの基礎, サイエンス社 (2007).