DD1366 Programmeringsparadigm
X1

Hugo Larsson Wilhelmsson, Erik Smit
hugolw@kth.se, esmit@kth.se
20020921-1234, 20020924-2437

15 mars 2024

1 Bus Numbers

1.1 Val av paradigm

Vi valde att använda det imperativa paradigmet samt det logiska paradigmet för att lösa denna uppgift. Anledningen till att vi valde det imperativa paradigmet (C++) för att lösa detta är eftersom vi vill utnyttja användandet av loopar, vilket inte finns i varken det logiska eller det funktionella paradigmet. Vi kan därmed iterera igenom samtliga bussnummer och genomföra separata operationer.

Anledningen till att vi också valde det logiska paradigmet är eftersom man kan formulera uppgiften som olika regler. Dessa regler anger vad som ska skrivas ut, t.ex när '-' ska skrivas. Vi valde här att först sortera listan med alla bussnummer, och sedan kolla de första två elementen för att se om de ska skrivas ut separat eller om endast det första ska skrivas ut och med hjälp av *check_when_same_in_row* ta reda på vilket element som ska skrivas ut sist och skriva '-' mellan dem. Implementationen i det logiska paradigmet använder sig dessutom av rekursion och basfall vilket lämpar sig bra för att gå igenom de olika reglerna.

1.2 Reflektion

Vi tycker att det var lättare att implementera en lösning i det imperativa paradigmet. Anledningen till detta är eftersom vi arbetat mycket mer med det imperativa paradigmet än det logiska paradigmet tidigare. När vi väl lyckades få grepp om hur vi skulle implementera lösningen i det logiska paradigmet så var det ganska tydligt vilka regler som behövdes, men den svåraste delen var att lista ut hur problemet skulle lösas.

2 Calculating Dart Scores

2.1 Val av paradigm

I denna uppgift valde vi att använda det funktionella paradigmet samt det imperativa paradigmet. Vi valde det imperativa paradigmet av samma anledning som i förra uppgiften, för att kunna utnyttja loopar. Looparna itererar genom de olika dart-zonerna och använder sig av metoden *get_multiplier* för att få fram de olika resultaten.

Att välja det andra paradigmet vi skulle använda till denna uppgift var svårare. Det slutade med att vi valde det funktionella paradigmet då vi tänkte att det skulle vara smidigare att använda selektioner för olika fall istället för regler i det logiska paradigmet. Det som sker i den funktionella implementationen är att vi använder funktionen *getScore* för att beroende på vilket resultat spelaren får ge det första resultatet. Därefter används funktionen *getOtherScores* för att antingen returnera resultatet eller printa en ny rad och sedan

anropa getScore igen. getScore kommer ge resultaten single 1-20, double 20 eller triple 20 och om något annat nummer behövs för att få det önskade resultatet så används funktionen *getSpecialScore* för att ta reda på och printa dessa.

2.2 Reflektion

Eftersom vi arbetat mycket med det imperativa paradigmet tidigare så var det enklare att implementera en lösning i det. Det underlättade också att vi kunde använda oss av loopar, vilket inte finns i det funktionella paradigmet. Det var däremot inte alltför svårt att lösa problemet i det funktionella paradigmet eftersom man kan använda sig av selektioner och därmed avgöra när vi skulle printa olika resultat.

3 Peragrams

3.1 Val av paradigm

Vi använde oss av det funktionella paradigmet och det logiska paradigmet för att lösa denna uppgift. Valet av det funktionella paradigmet grundade sig i att vi ville använda selektioner och rekursioner. Vi använder till exempel funktionen *minNum* för att kolla basfall och i vissa fall returnera 0, annars vill vi anropa funktionen *checkOddTuples*. Denna funktion tar in tupler och tar reda på antalet karaktärer som behöver tas bort för att bilda ett Peragram, med hjälp av rekursion.

Valet av det logiska paradigmet grundar sig i att vi ville sätta upp olika regler för att avgöra hur många karaktärer som behöver tas bort för att bilda ett Peragram. Till exempel så är en regel att om längden av ordet är jämnt och inget tecken förekommer udda antal gånger så skrivs “0” ut. Detsamma gäller om längden av ordet är udda men att det endast finns ett tecken som förekommer udda antal gånger. Om inget av dessa reglerna gäller, så kommer den sista regeln gälla, som resulterar i att den printa antalet tecken som förekommer udda gånger - 1.

3.2 Reflektion

Vi förstod ganska snabbt hur detta problemet matematiskt kan lösas, och ganska snabbt lyckades vi lösa uppgiften i det funktionella paradigmet. Anledningen till detta var att vi kunde använda rekursion och basfall. Vi översatte inte koden rätt av när vi sedan löste den i det logiska paradigmet, däremot så använde vi oss av samma matematiska tänk med att kolla antalet gånger karaktärer förekommer och längden av ordet. Överlag tycker vi att det var jobbigare att lösa uppgiften i det logiska paradigmet eftersom vi inte jobbat med det lika mycket tidigare och tycker det är svårare att formulera regler än rekursiva funktioner.

4 Slutgiltig reflektion

Vi tyckte att det var en rolig laboration. Vi insåg hur mycket smidigare det kan vara att lösa uppgifter i det imperativa paradigmet om det går att använda sig av till exempel loopar. Vi insåg däremot också hur smidigt det kan vara att använda sig av rekursion om man vill lösa liknande problem i det funktionella paradigmet. Det logiska paradigmet har sin charm med att formulera olika regler, som kan underlätta till exempel i matematiska problem men eftersom detta är ett paradigm som känns mer annorlunda än andra paradigm så tyckte vi att detta var svårare. Vi anser därmed att majoriteten av problemen som kan lösas i det logiska paradigmet även kan lösas i andra paradigm