



Title: Bakery Management System Documentation

Intake code: UCDF2309ICT(SE)

Group Number: Group PWPAB

Module Name: Programming with Python

Module Code: AAPP015-4-1-PWP

Lecturer: Ms. Aziah binti Abdollah

Group Member	TP Number
Ng Yvonne	TP076390
Lum Han Xun	TP076160
Heng Xin Hui	TP077232

Table of Contents

1.0 Introduction	5
2.0 Assumptions.....	6
2.1 Manager	6
2.2 Customer.....	7
2.3 Cashier	8
2.4 Baker	9
3.0 Flowchart.....	10
3.1 Main	10
3.2 Manager	11
3.2.1 System Administration	14
3.2.2 Order Management	36
3.2.3 Financial Management	38
3.2.4 Inventory Control	47
3.2.5 Customer Feedback	64
3.2.6 Notifications.....	66
3.3 Customer.....	71
3.3.1 Customer Account Management	71
3.3.2 Product Browsing.....	86
3.3.3 Cart Management	89
3.3.4 Order Tracking.....	97
3.3.5 Product Review	98
3.3.6 Customer Loyalty Rewards	99
3.5 Baker(s).....	102
3.5.1 Equipment Management	104
3.5.2 Product Keeping.....	111
3.5.3 Recipe Management.....	124

3.5.4 Inventory Check	140
3.4 Cashier	143
3.4.1 Product Display	145
3.4.2 Manage Discount	146
3.4.3 Transaction Completion	151
3.4.4 Reporting	154
4.0 Concept of code snippets	165
4.1 Auxiliary function	165
4.2 Main	172
4.3 Manager	174
4.3.1 Sign up / login page	174
4.3.2 System Administration	176
4.3.3 Order Management	199
4.3.4 Financial Management	201
4.3.5 Inventory Control	207
4.3.6 Customer Feedback	219
4.3.7 Notifications	221
4.4 Customer	225
4.4.1 Customer Account Management	225
4.4.2 Product Browsing	234
4.4.4 Order Tracking	245
4.4.5 Product Review	247
4.4.6 Customer Loyalty Rewards	250
4.5 Cashier	256
4.5.1 Sign In/ Log In	256
4.5.2 Product Display	258
4.5.3 Discount Management	260

4.5.4 Transaction Completion	265
4.5.5 Reporting	268
4.6 Baker(s).....	281
4.6.1 Sign Up/ Log In	281
4.6.2 Equipment Management	283
4.6.3 Inventory Check.....	293
4.6.4 Product Keeping.....	297
4.6.5 Recipe Management.....	313
5.0 Conclusion	337
6.0 References	338
7.0 Workload matrix.....	339

1.0 Introduction

The Morning Glory Bakery Management System is designed to make it easier for a small family-owned bakery to manage its daily tasks. Currently, the manual methods used for various processes can be time-consuming and have high risk of mistakes and these have made it difficult for the bakery to provide good service and grow effectively (Anon, 2019). Therefore, the main goal of our project built in Python is to simplify the main operations of the bakery by creating a complete system. By automating important processes, our bakery system can hopefully improve overall workflow efficiency and customer satisfaction could also be boosted since it better meets their demands through innovation. The Bakery Management System should handle daily tasks smoothly to create a satisfying experience for customers and also enhance accuracy (MuftaSoft, 2023). Our Morning Glory Bakery System also aims to enable all of the managers, cashiers and bakers carry out their tasks effectively in order to improve customer service and productivity while customers can also have a good experience.

2.0 Assumptions

To ensure the system function correctly, there are some assumptions that need to be considered. First of all, it is assumed that Morning Glory Bakery is a small, family-owned business, operate for almost 1.5 years, that have 1 manager, 2 bakers and 1 cashier. To access to the bakery management system, they are required to verify their identity to avoid unauthorized access that may cause unexpected system failure such as disclosure of customer information. All data that save to files will go through the validation process if necessary to ensure their accuracy. For instance, expiry date, contact number and email address.

2.1 Manager

It is assumed that manager has the ability to manage all roles including bakers, cashiers and customers. Manager needs to log in to the system with a specific password to access the privileges. He/she is responsible to create accounts for bakers and cashiers, set a default password for them to access respective privileges. It can be changed later by bakers or cashiers, and the changes will be recorded into the system. Therefore, manager can access to all privileges. The age of bakers and cashiers to be added should be between 17 and 61.

In the system administration, it is assumed that a baker or a cashier can only have one account. The manager can add baker and cashier accounts, remove accounts and update their information. If there is left only one baker, the account cannot be removed to ensure daily operations. Since it has only one cashier, if the manager wants to remove the cashier, it is assumed that he/she wants to hire another person to be the cashier. While for the customer accounts, manager can update and terminate customer information if customers are facing problems to update or terminate themselves. Once the customers have been terminated, they need to sign up again to access to the system. The manager can also activate and deactivate their accounts based on their log in activities. For example, manager will deactivate the user account if he/she has not been logged into the system for an extended period.

It is assumed that manager can view and update the order details of customers. When customers place their orders, their order status will change to ‘Order Placed’. The manager can monitor the order status in real time, to check which orders have not been completed. Assume that customers have complete their payment, manager will update the order status to ‘Payment

Completed'. If customers want to cancel their orders, manager can also change the status to 'Canceled'.

In the financial management, manager is able to track income, expenses and profitability of the bakery. Manager can view monthly or yearly net profit and profit margin to monitor the business growth. It is assumed that the net income to calculate net profit has exclude the miscellaneous (rental, water and electricity bill, salary and so on).

Manager can add or delete stock of products in the inventory or update the information of the products. Manager is assumed to add the price and description of the products for menu display while adding the stock. Besides, it is assumed that manager is responsible to buy the ingredients and add to the inventory timely to avoid disruption in the supply chain that delay the production of bakers, which might result in a product shortage. The ingredients will be classified into 10 main categories. Each category has the specific form and unit measurement. Manager is assumed to remove the ingredients from the inventory if the ingredient is expired or mouldy.

Furthermore, manager will receive notifications of malfunction equipment and equipment that need maintenance from bakers. It is assumed that if the equipment is repair by manager or bakers themselves and fail to function well, they are unable to claim the warranty because it is human-made damage unless they ask the manufacturer to repair directly. On the other hand, if they ask the manufacturer to repair the equipment, no matter it is function well or not after the repair, they are eligible to claim the warranty from the manufacturer. Manager is responsible for keeping a record for the equipment repair.

It is also assumed that manager can view the customer feedback to step in their shoes, helps in enhancing the customer services and business of the bakery in the future. Meanwhile, manager can response to customer feedback, giving them confidence that their feedback has been read and considered.

2.2 Customer

Every customer in the Bakery Management System is expected to have a username that acts as their own unique identifier. Customers can manage their accounts at any time through the system such as updating personal information or deleting their accounts.

Besides, customers can browse the full menu of the bakery and select products by category to help narrow down their options and save time. After placing an order, it is expected that customers will have the opportunity to provide their feedback and suggestions based on their experience. Also, it is assumed that customers will be able to track their orders after completing the payment. Customer can enter the order ID to view the status of their order and understand its current progress.

In addition, customers should be able to earn loyalty points based on their total spending in the bakery and understand the benefits of accumulating points such as using earned loyalty points to redeem cash vouchers. Hopefully, customers are expected to interact positively with the bakery system and have a good and engaging user experience.

2.3 Cashier

For the product display feature, it is assumed that cashiers have the ability to change the structure or layout of the product menu and can modify the data displayed in the product menu for all products. Besides, cashiers are expected to manage item discounts effectively such as adding discounts for newly launched items, deleting or modifying discounts and viewing all current discounts easily. It is assumed that if the customer request for the receipt of their previous purchase, cashier can manually generate the receipt. Otherwise, the receipt will always be generated after each purchase.

Lastly, cashiers are able to generate and save sales performance reports for the bakery to track sales performance. There are two types of sales reports: yearly and monthly. Cashiers can only generate reports for available years or months that exist in the customer transaction history. Additionally, cashiers can generate product popularity reports to find out the top and least-selling products within specific time periods, which are yearly, monthly, and an overall summary. Similar to sales performance reports, cashiers can only generate reports for available years or months that exist in the customer transaction history. The product popularity report provides insights into average ratings and customer reviews of both top and least-selling products. Customer ratings are also retrieved based on the selected period, according to the customer order date.

2.4 Baker

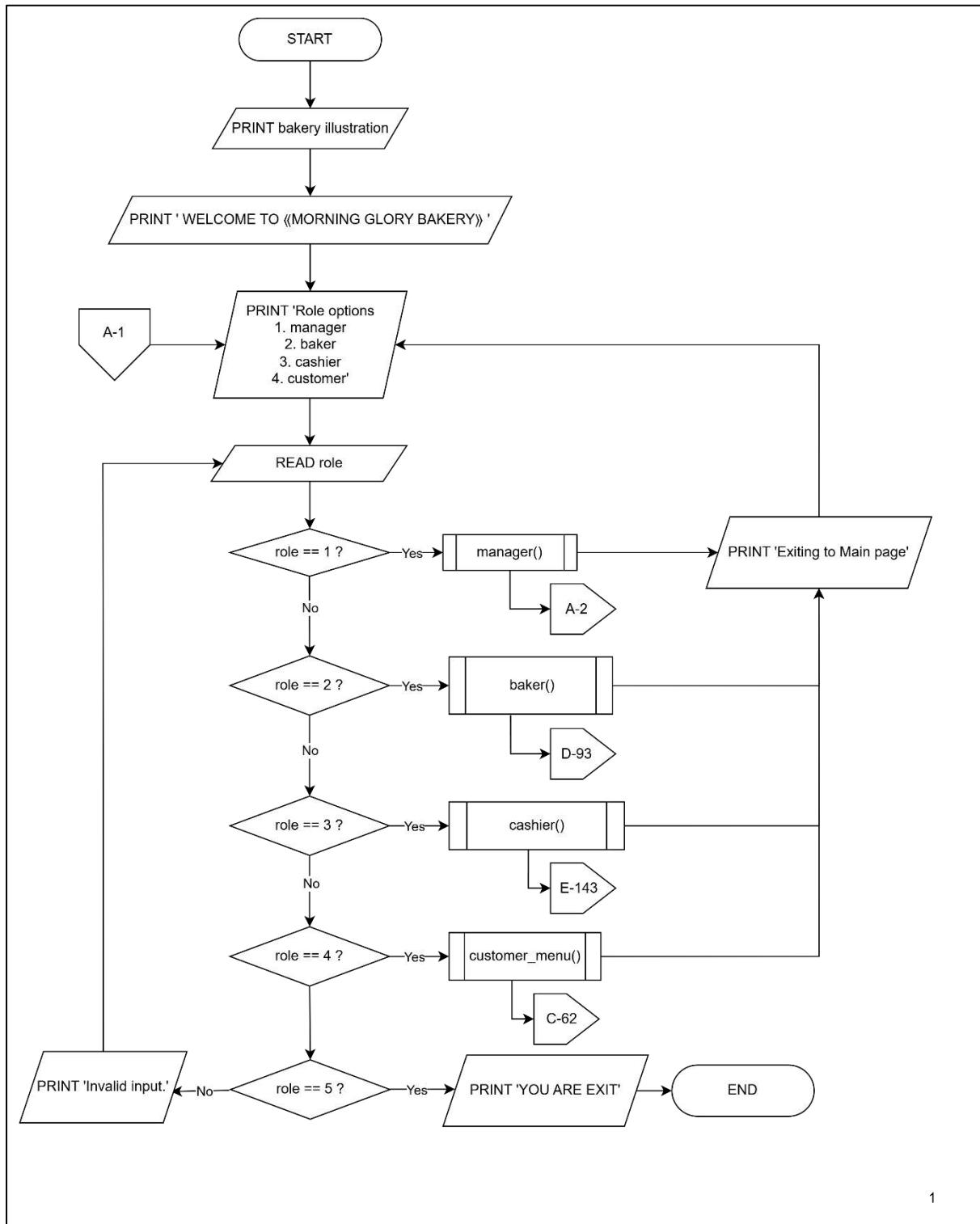
Baker is assumed to have ability to report malfunction and maintenance needs to manager if any such incidents occur. Baker needs to submit a report with the details of incident such as date of incident, issue description, severity level and so on, so that the manager can prioritize tasks and schedule maintenance. Furthermore, baker can create, update and delete recipes. To add a recipe, baker is required to enter details such as category, name, ingredient used, equipment used, baking information, instructions and cost per unit. Baker can only select from the available ingredient and equipment retrieved from the manager's inventory file and equipment file. Baker can update all attributes of the recipe data and delete a recipe when it is unnecessary.

Baker can also check ingredient availability and manage production. Baker needs to select recipe to work on from existing recipes and enter the quantity to produce. The inventory check begins by verifying whether the current ingredients in manager's inventory are available for the requested production quantity. Unit conversions are also handled if necessary (for example, kg to g, g to kg, l to ml, ml to l and more). If all ingredients are available, the quantities of ingredients in inventory will be subtracted by the quantities used, and production details will be saved.

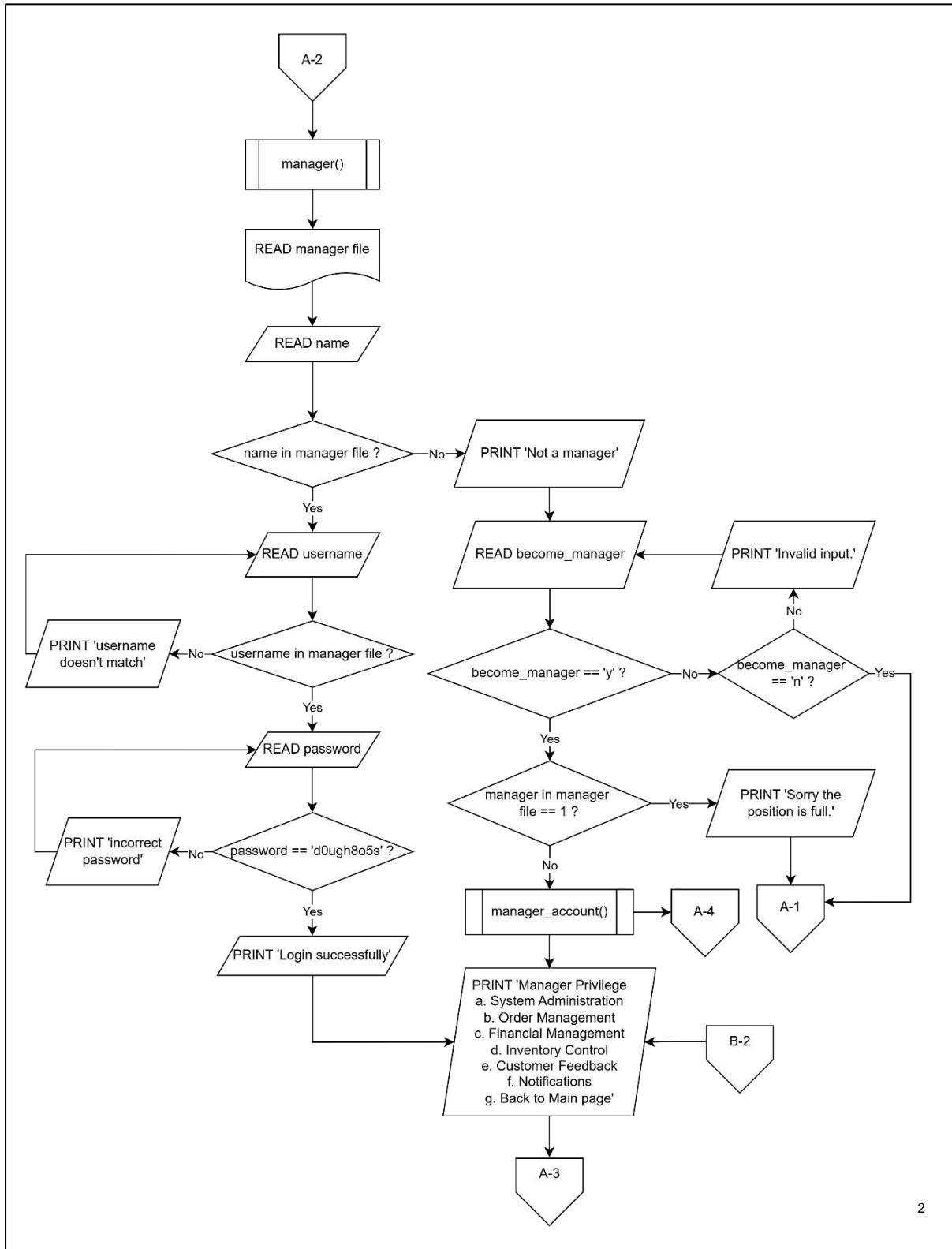
Lastly, baker is responsible for recording the details of products produced. When adding a product record, the date of production, category and production quantity must match the unsaved product details. Every product recorded by the bakers has a unique serial number and the products produced at one time share the same batch number. Each type of product is also represented by a product code that will be displayed in product menu for user selection. There is also a limit on the shelf life based on the product category. Baker cannot set an expiry date beyond the sum of the production date and shelf life entered plus one day. Baker can edit all attributes of recorded product except the date of production, production quantity and category. Baker can delete any product record if necessary.

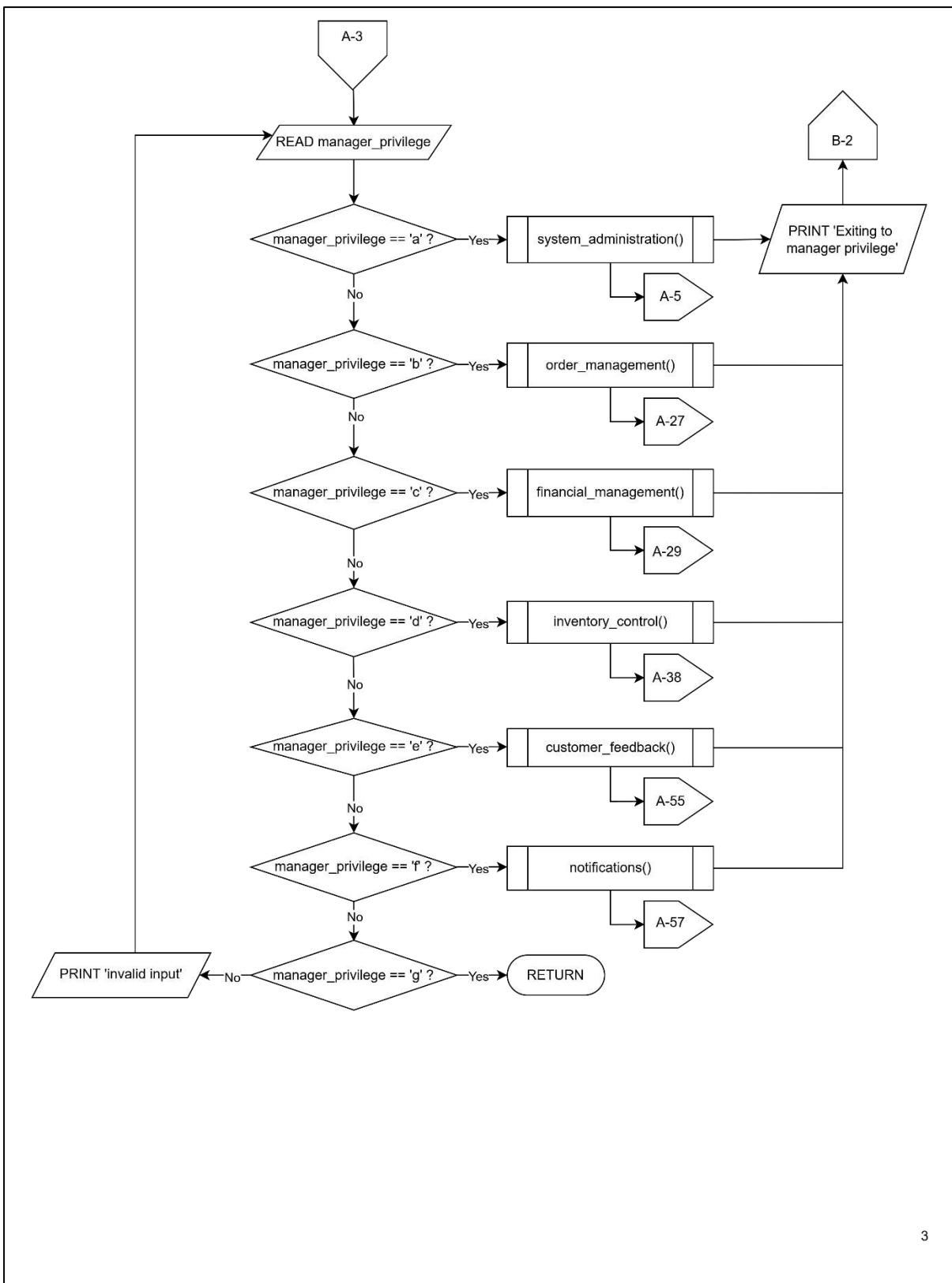
3.0 Flowchart

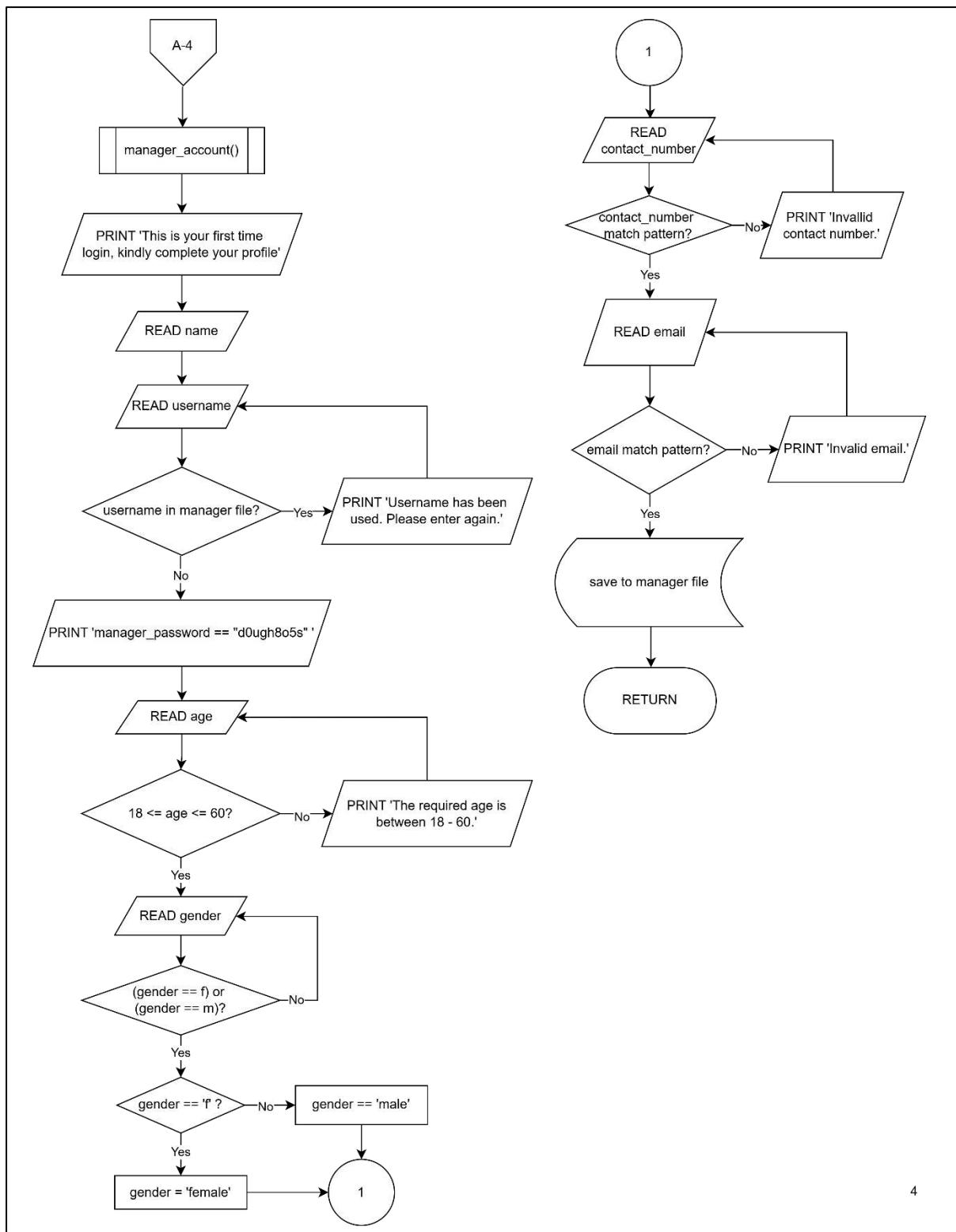
3.1 Main



3.2 Manager

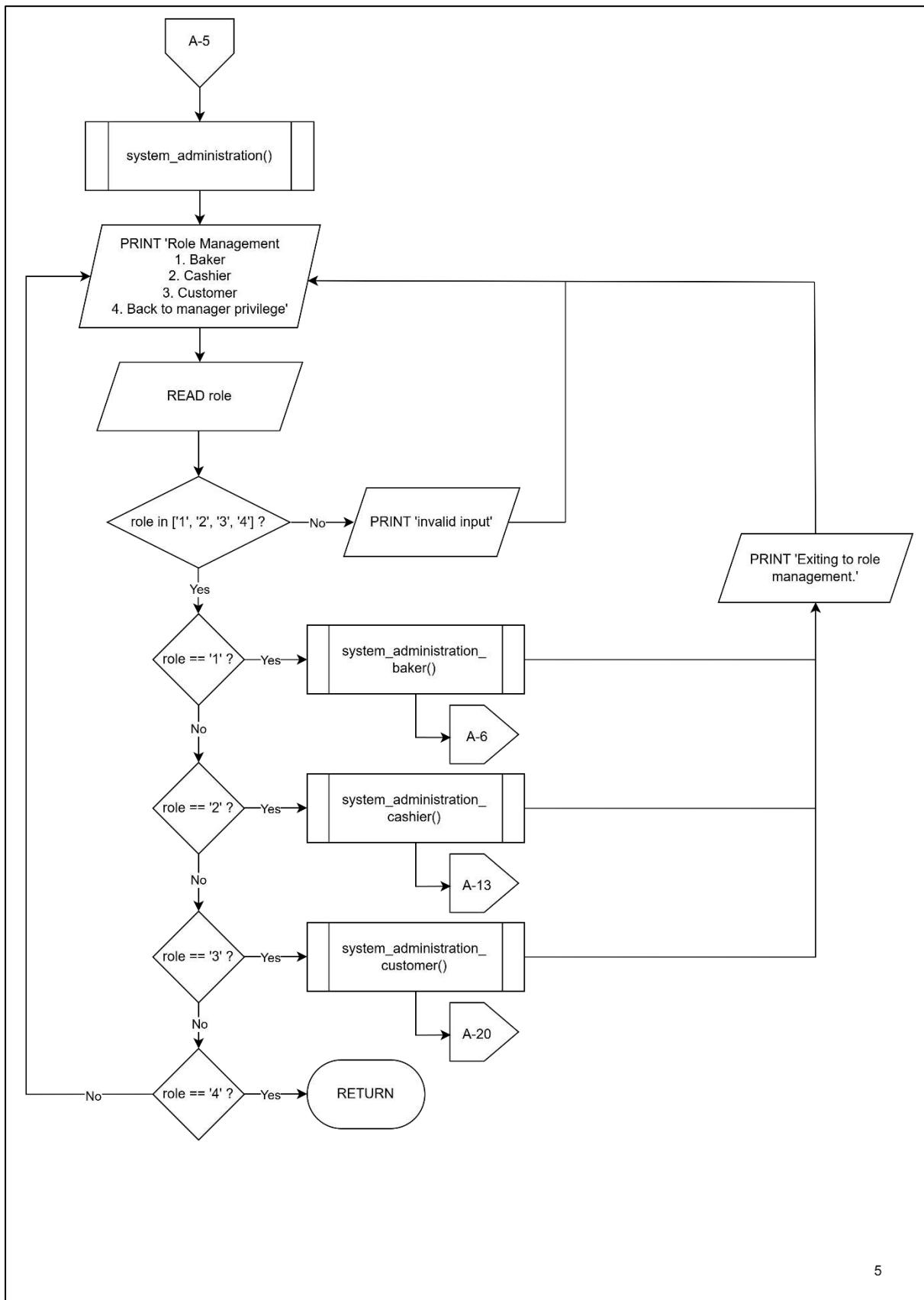


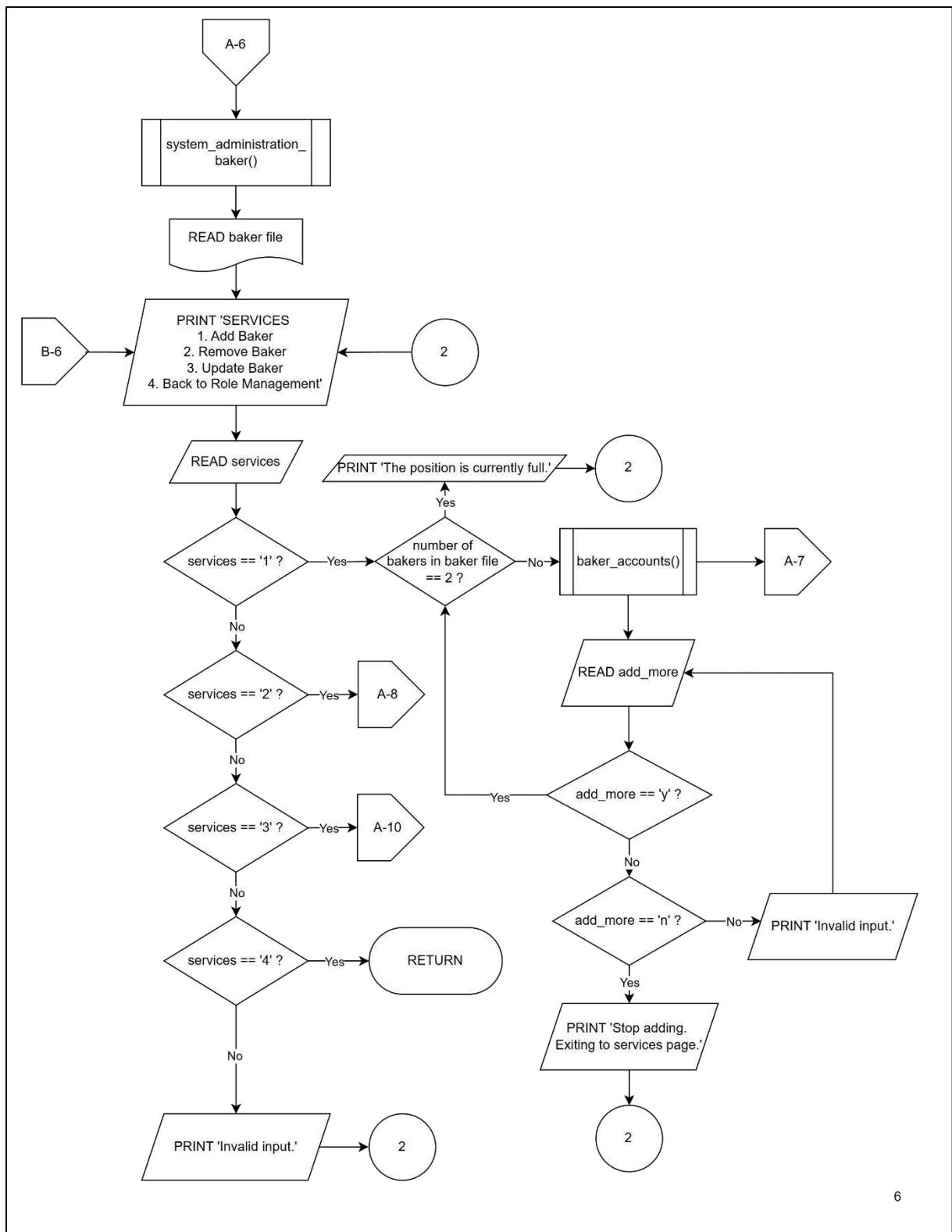




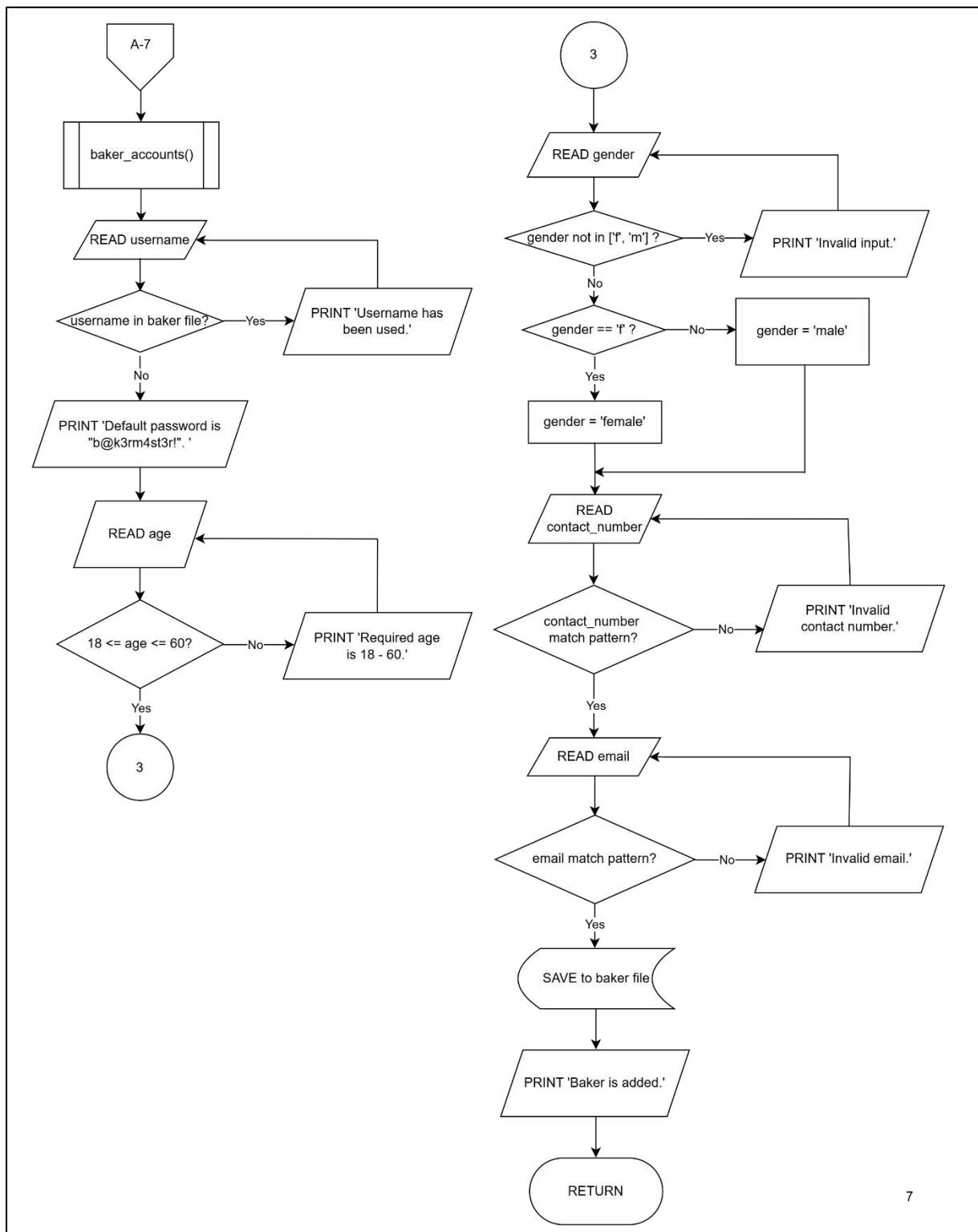
4

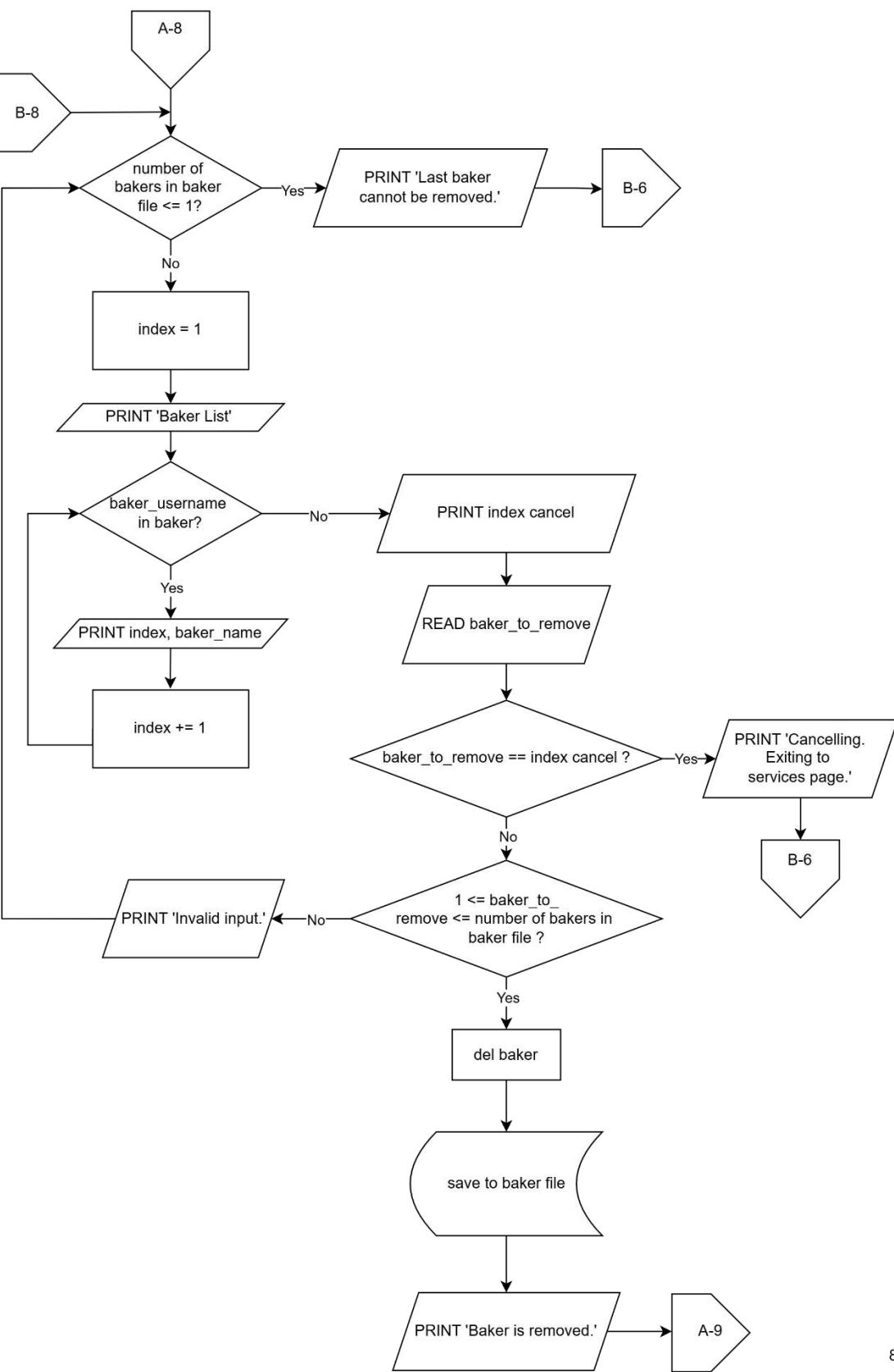
3.2.1 System Administration



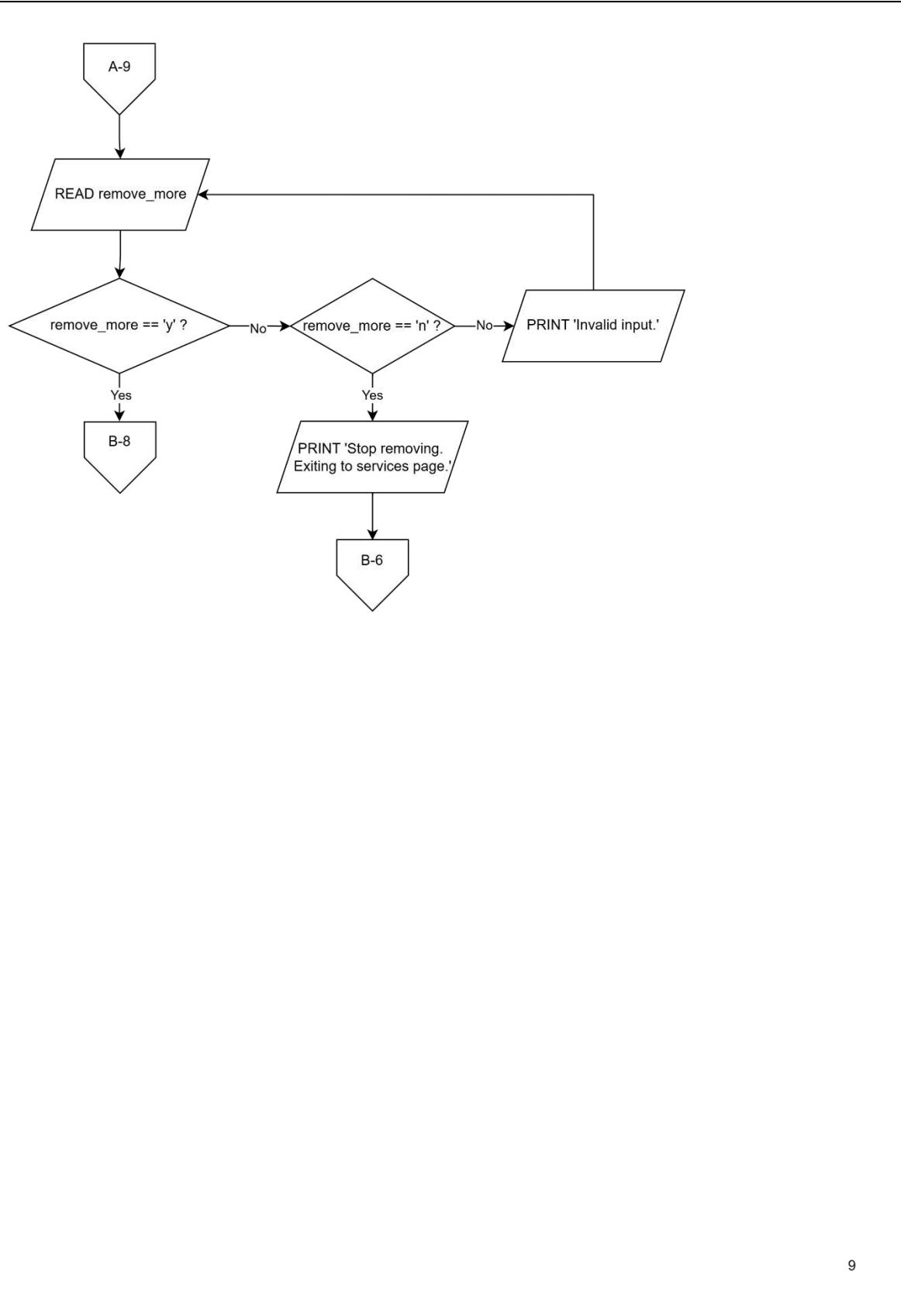


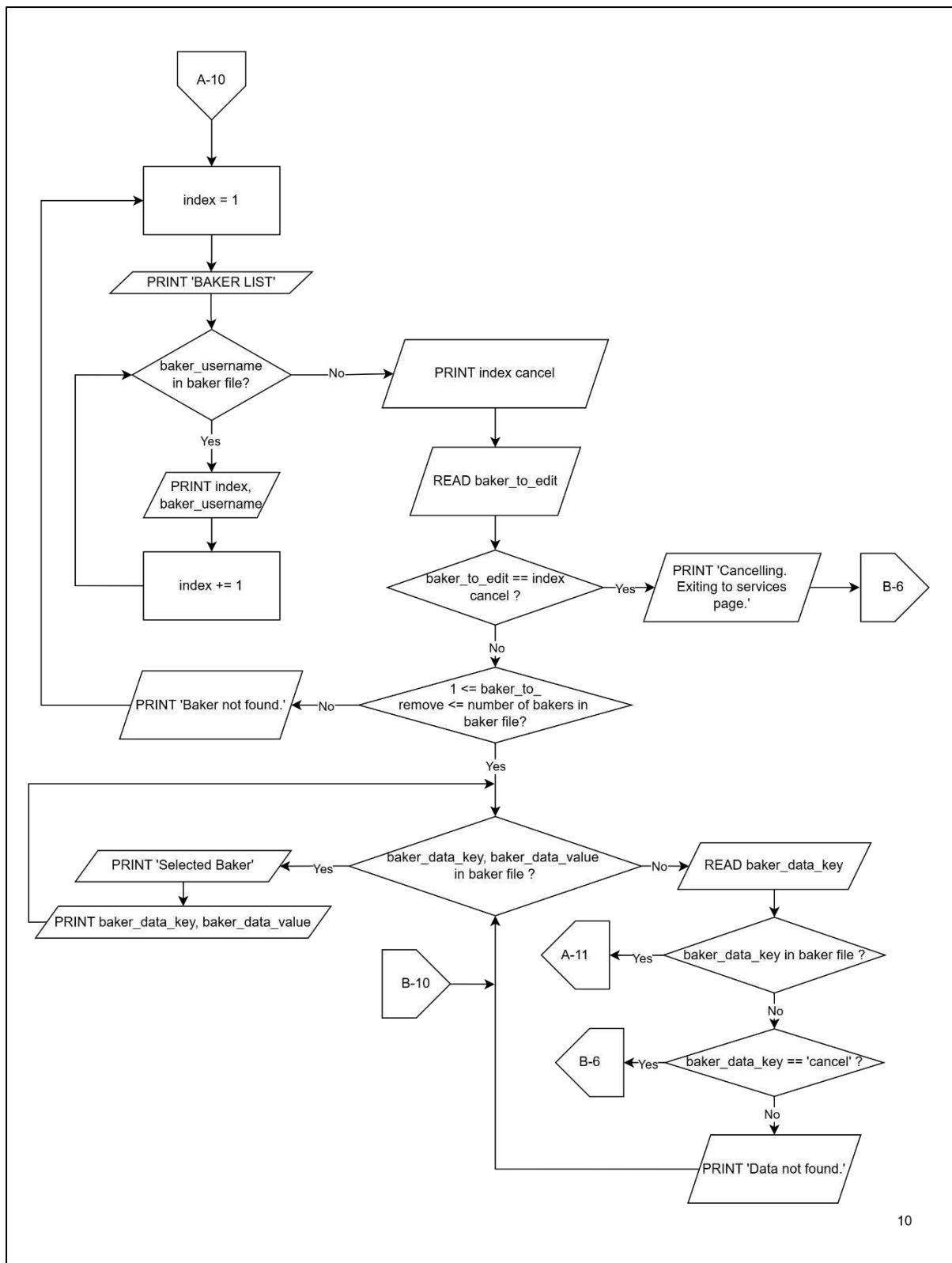
System administration of bakers



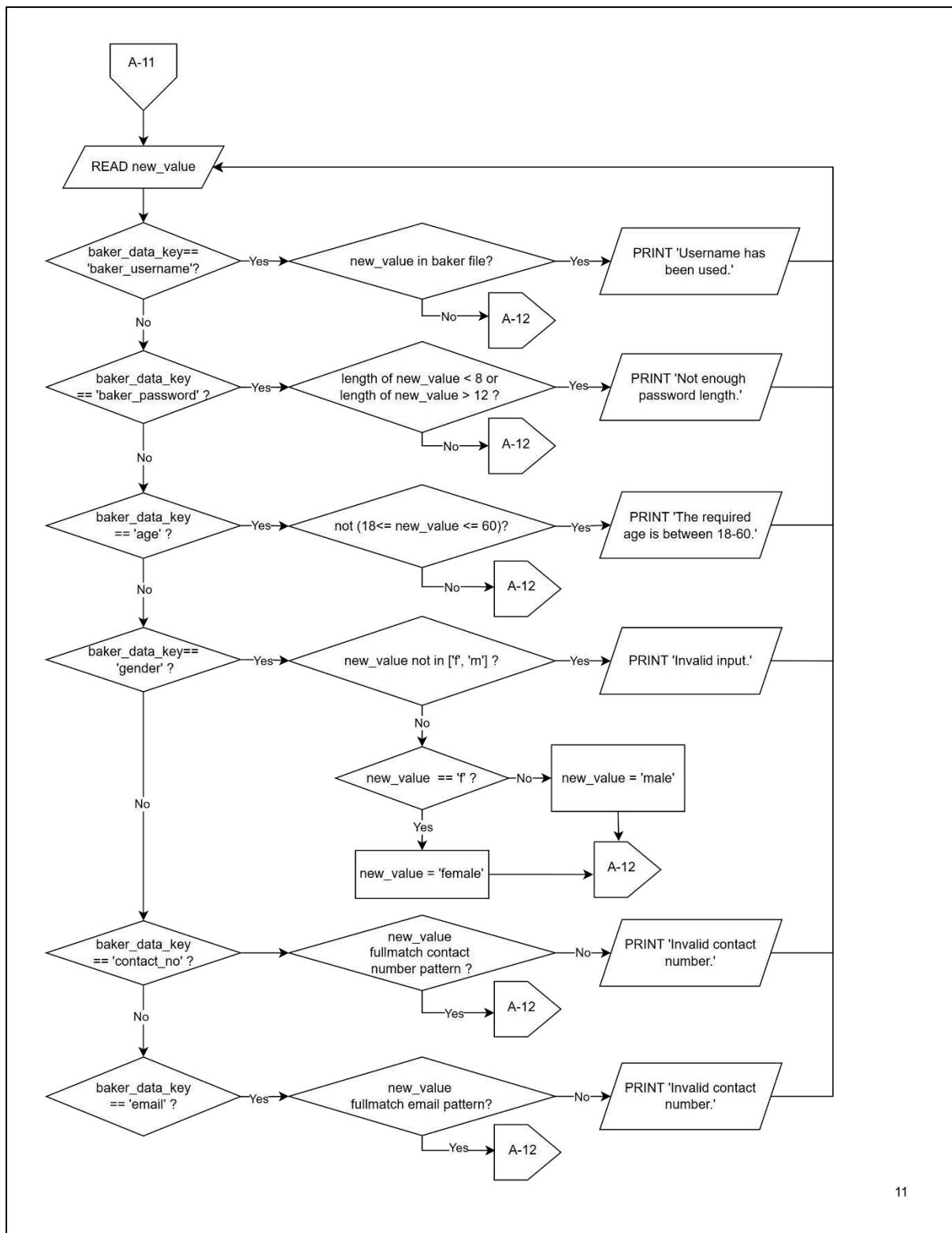


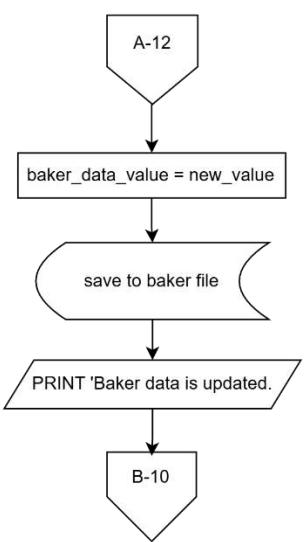
8





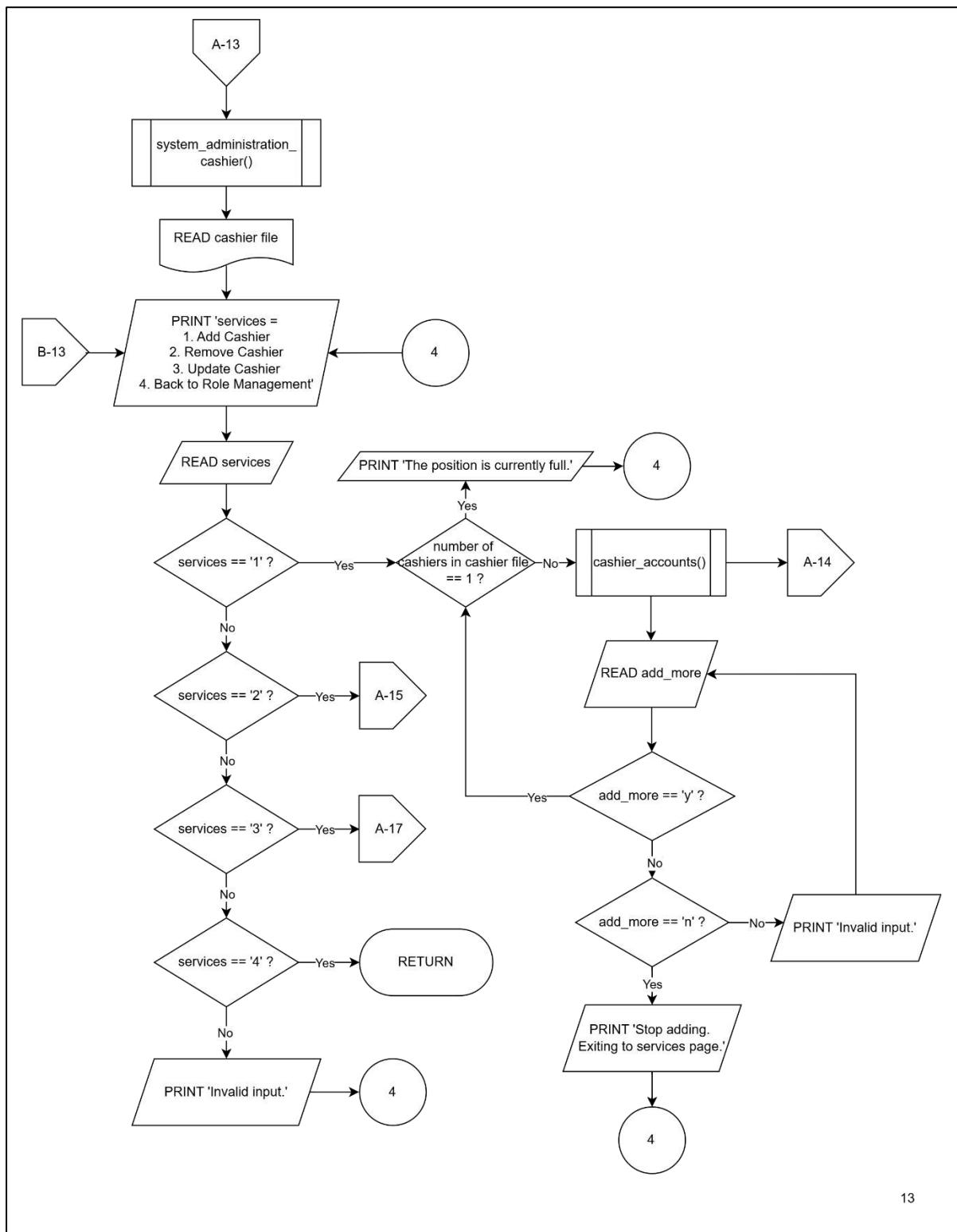
10

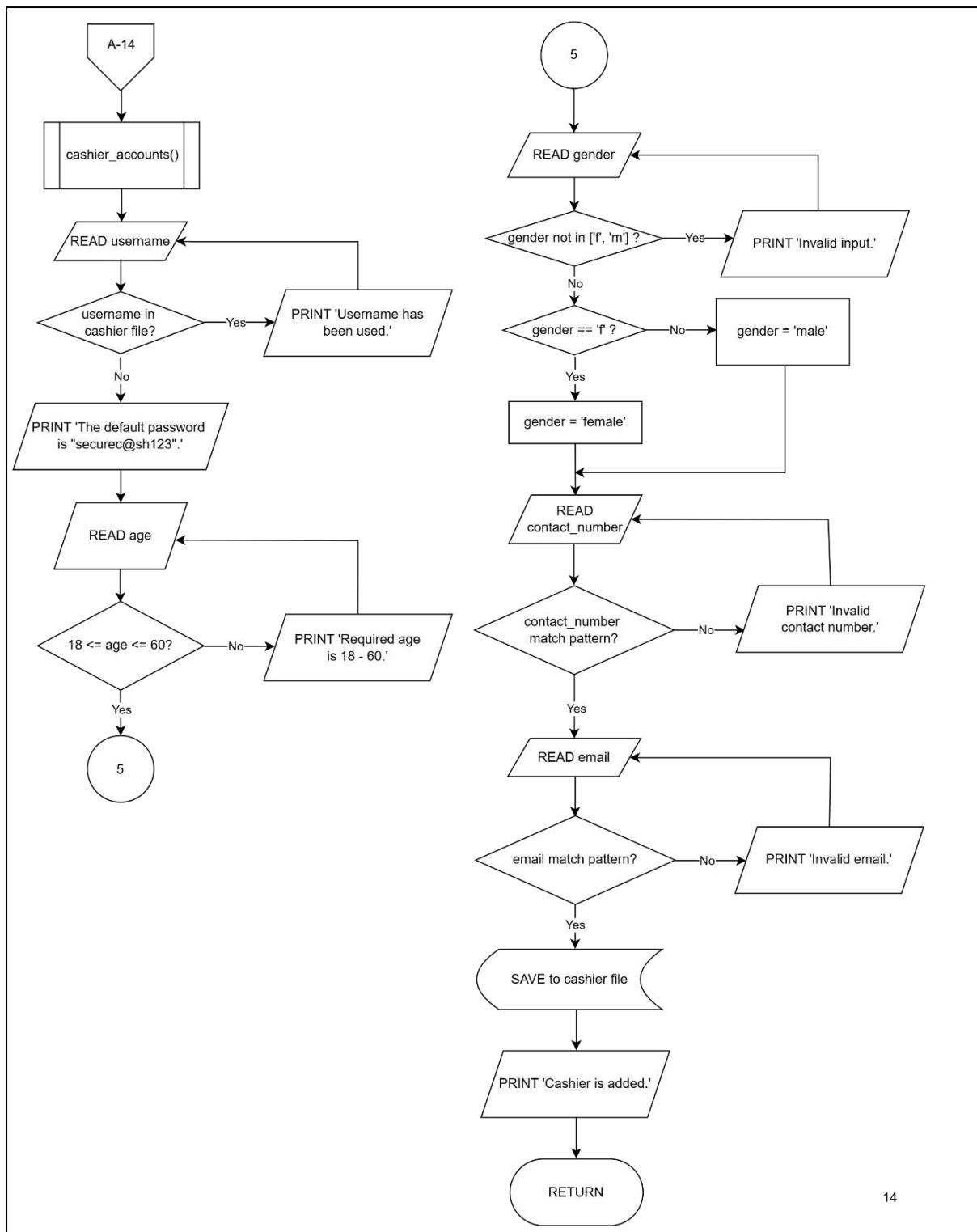




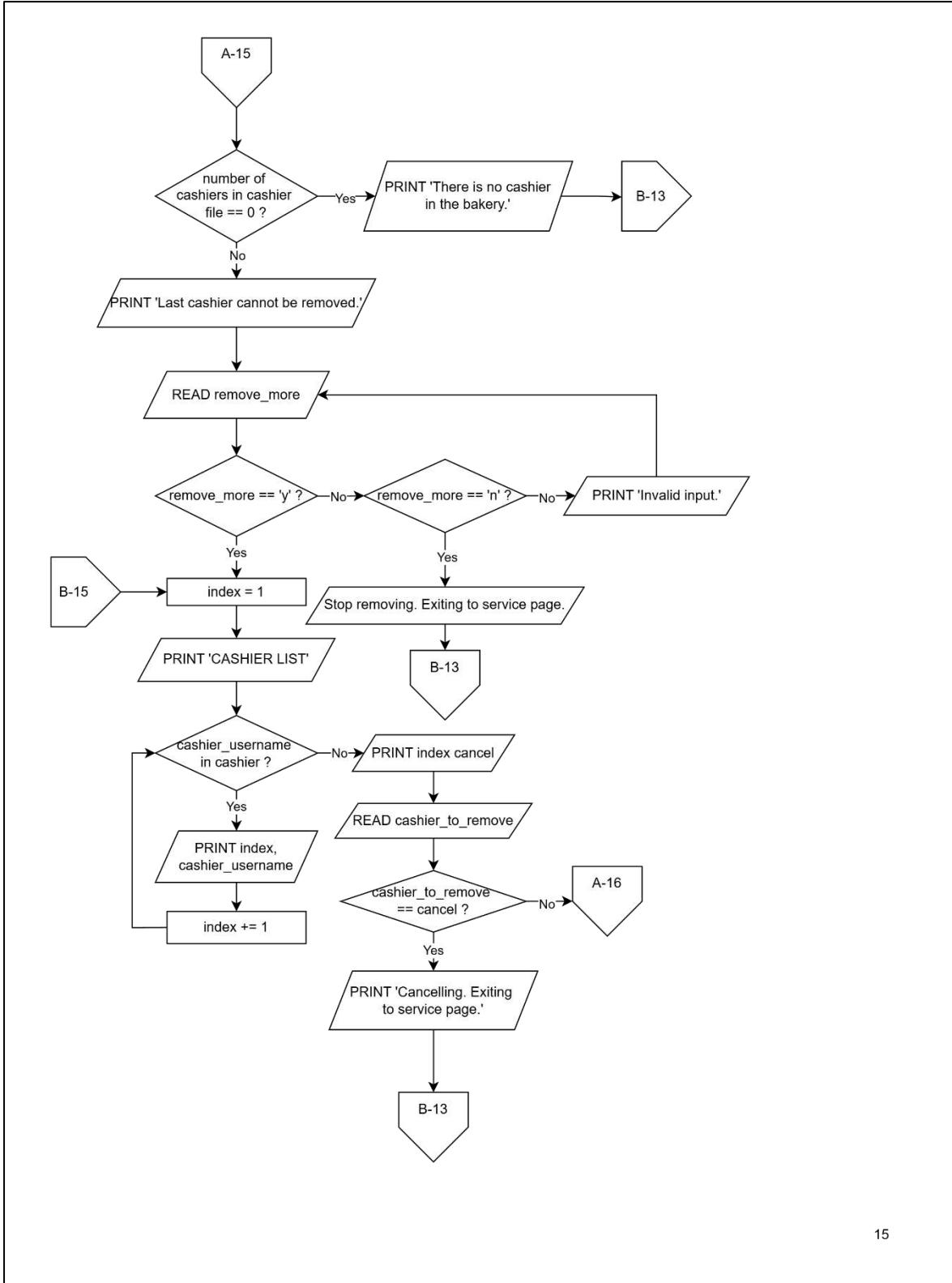
12

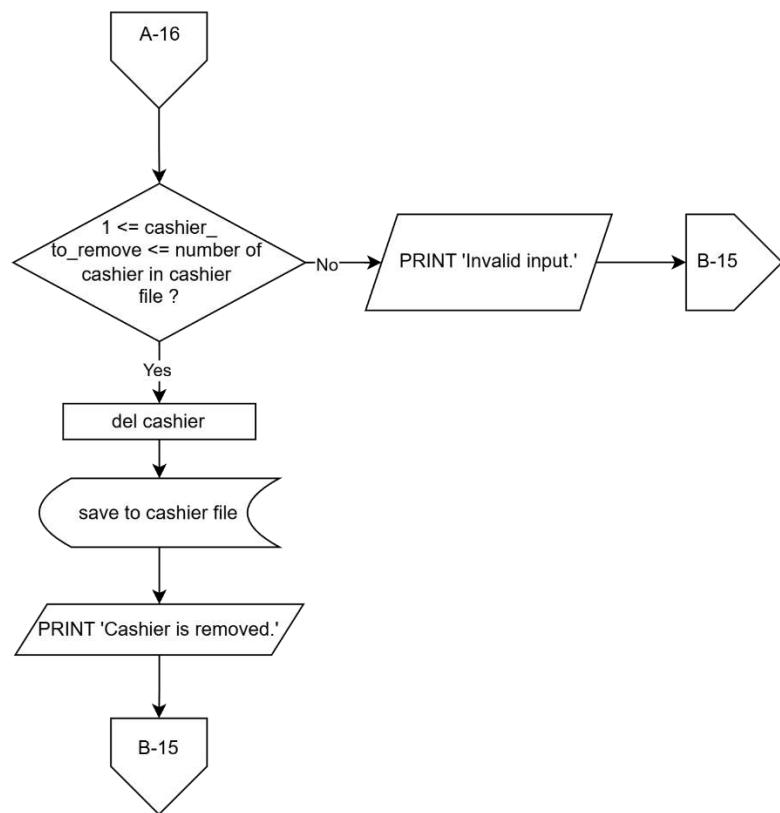
System administration of cashier

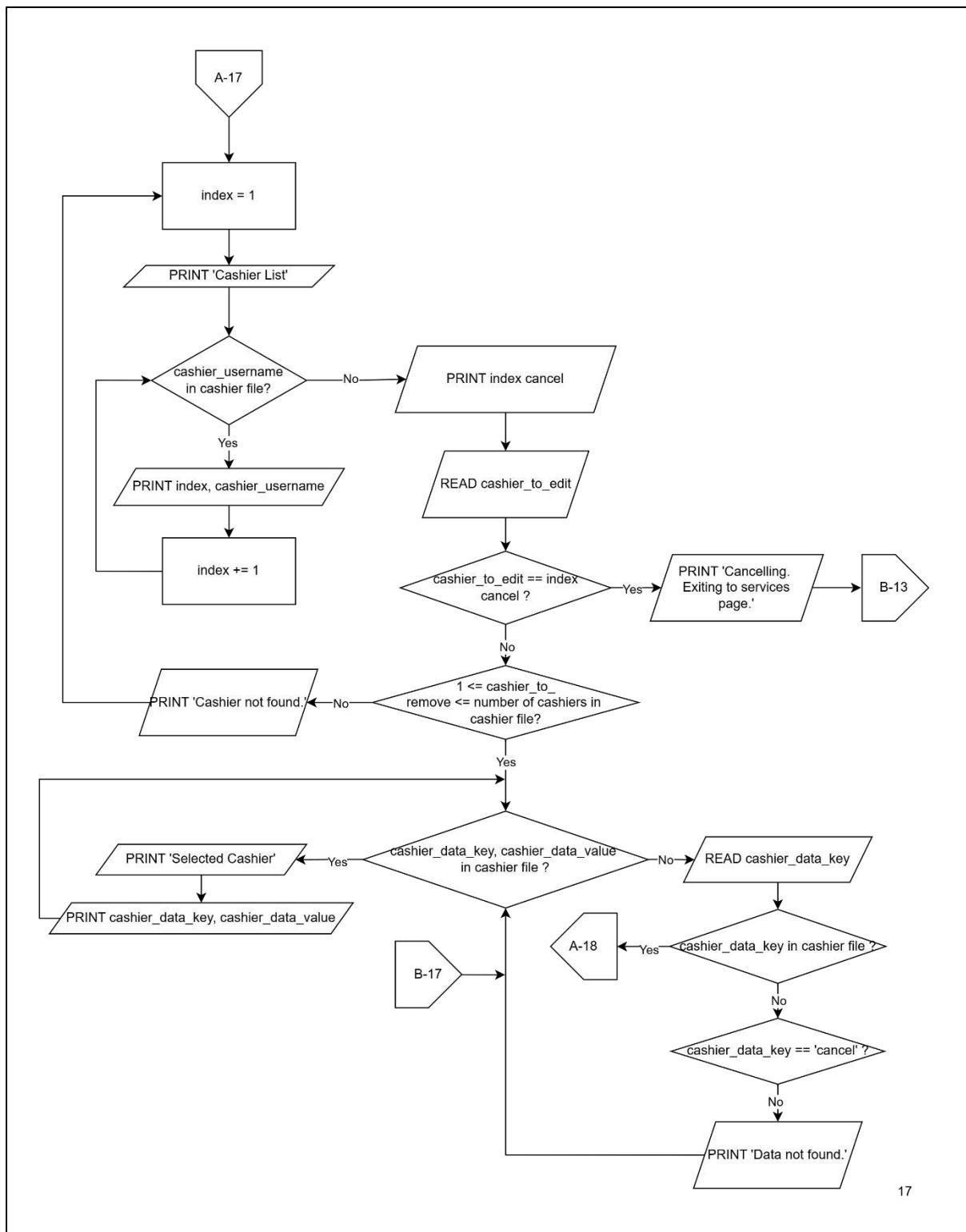




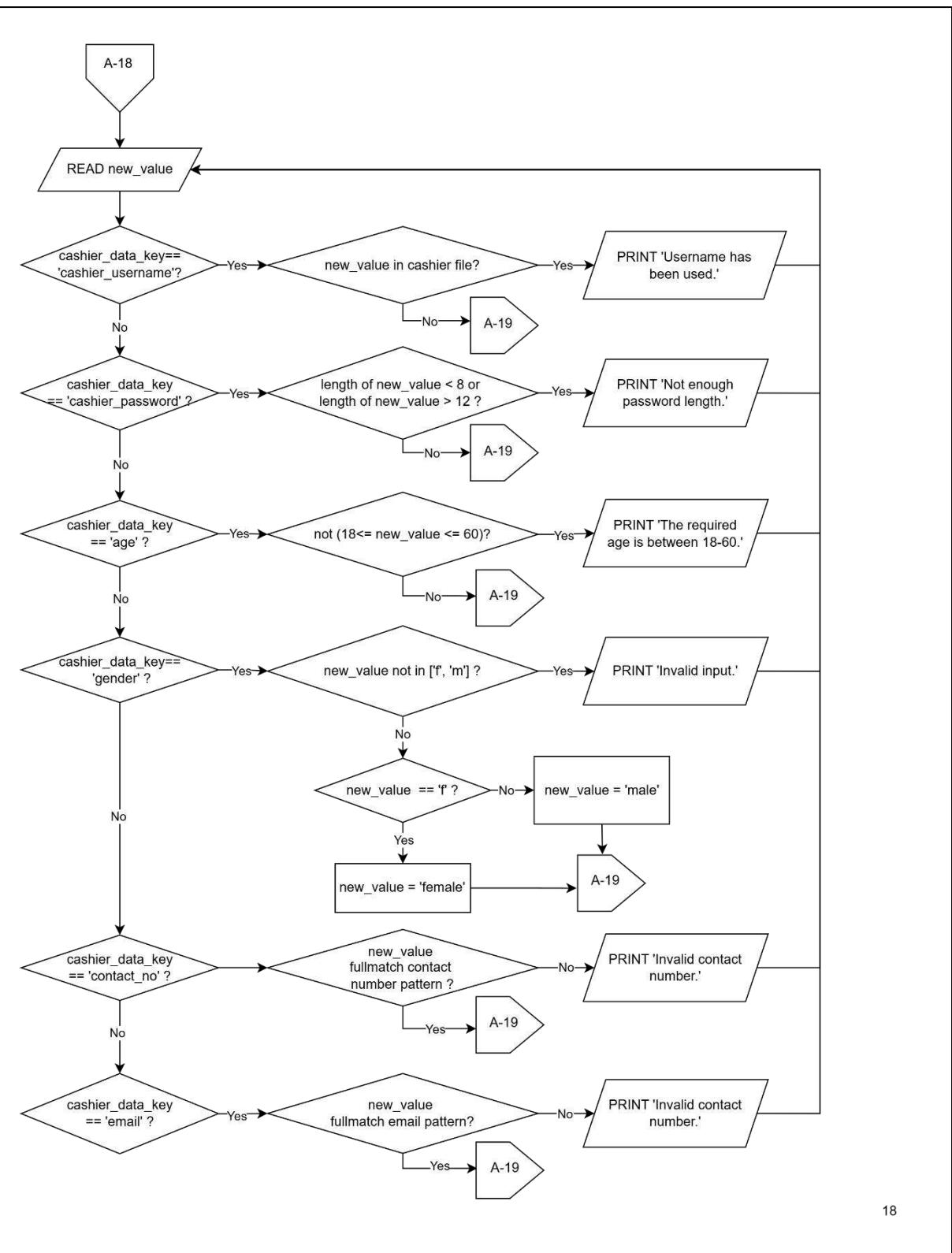
14

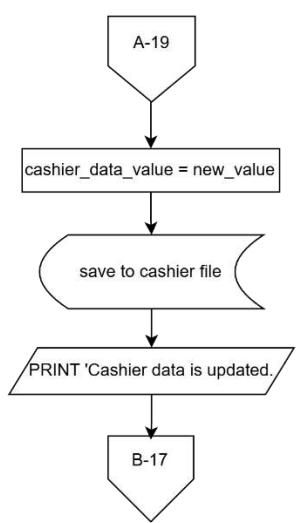




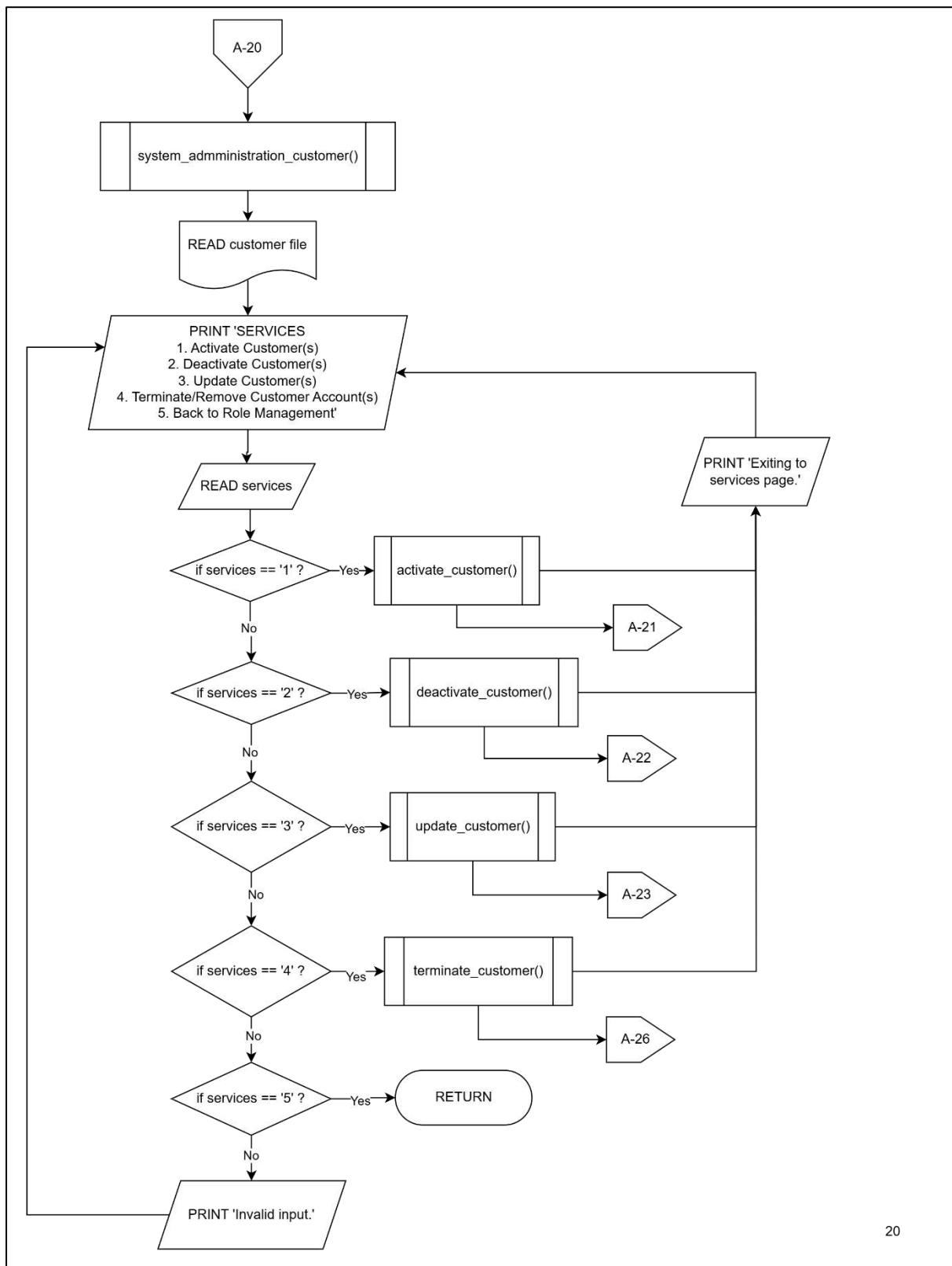


17

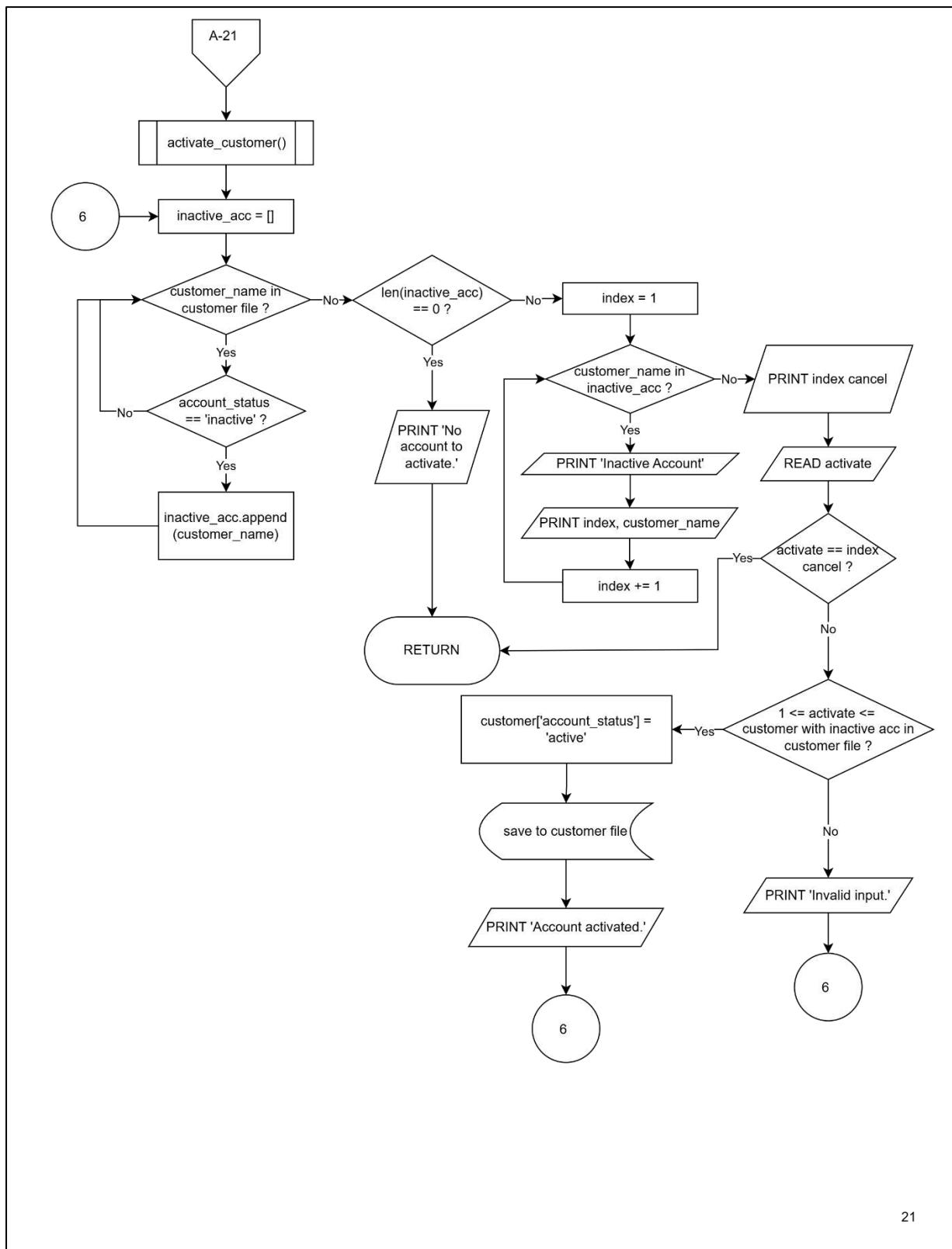


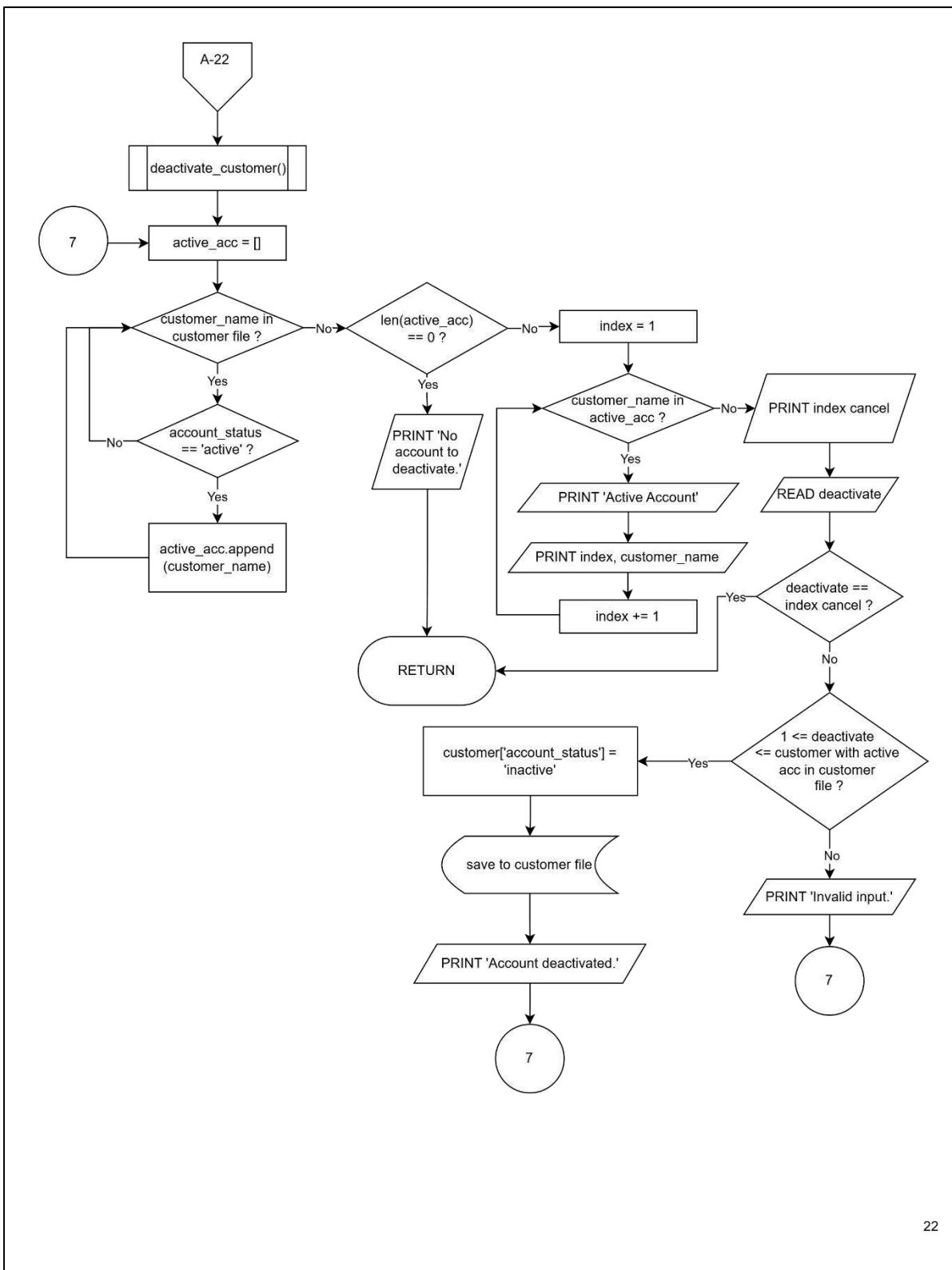


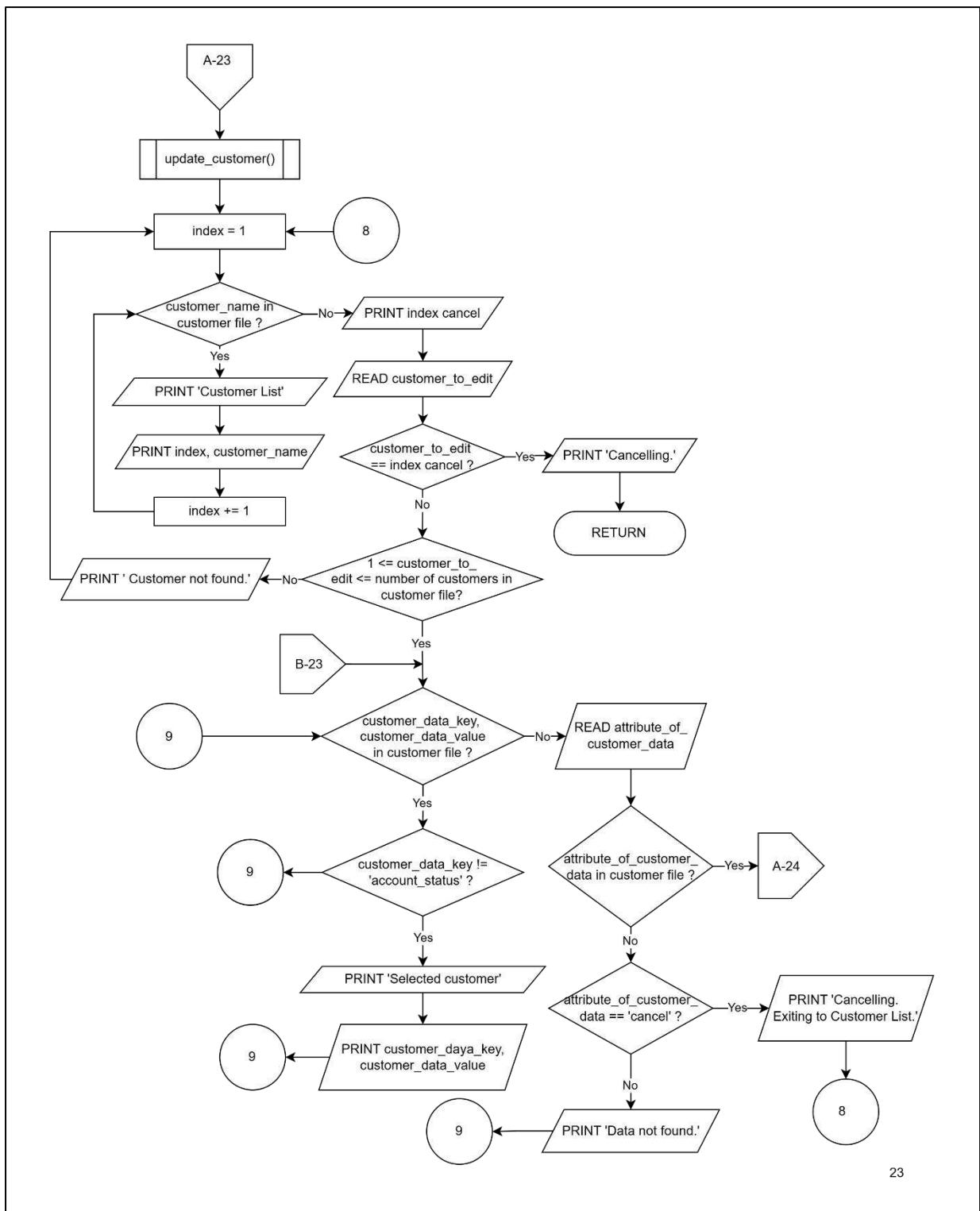
System administration of customers



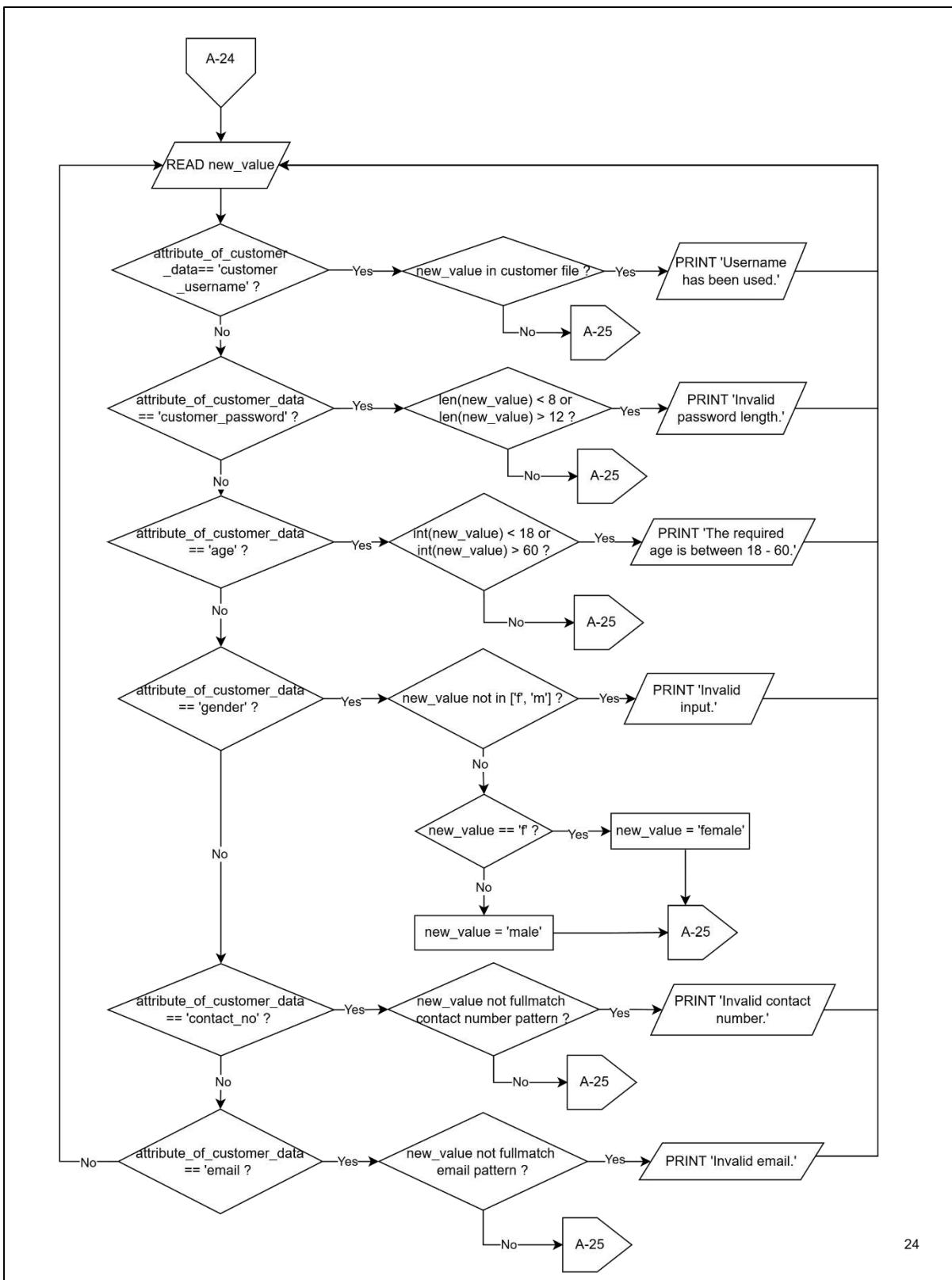
20



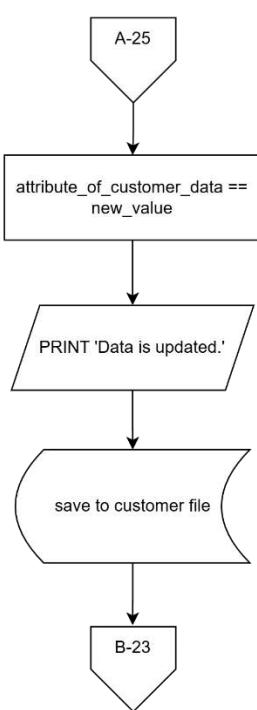




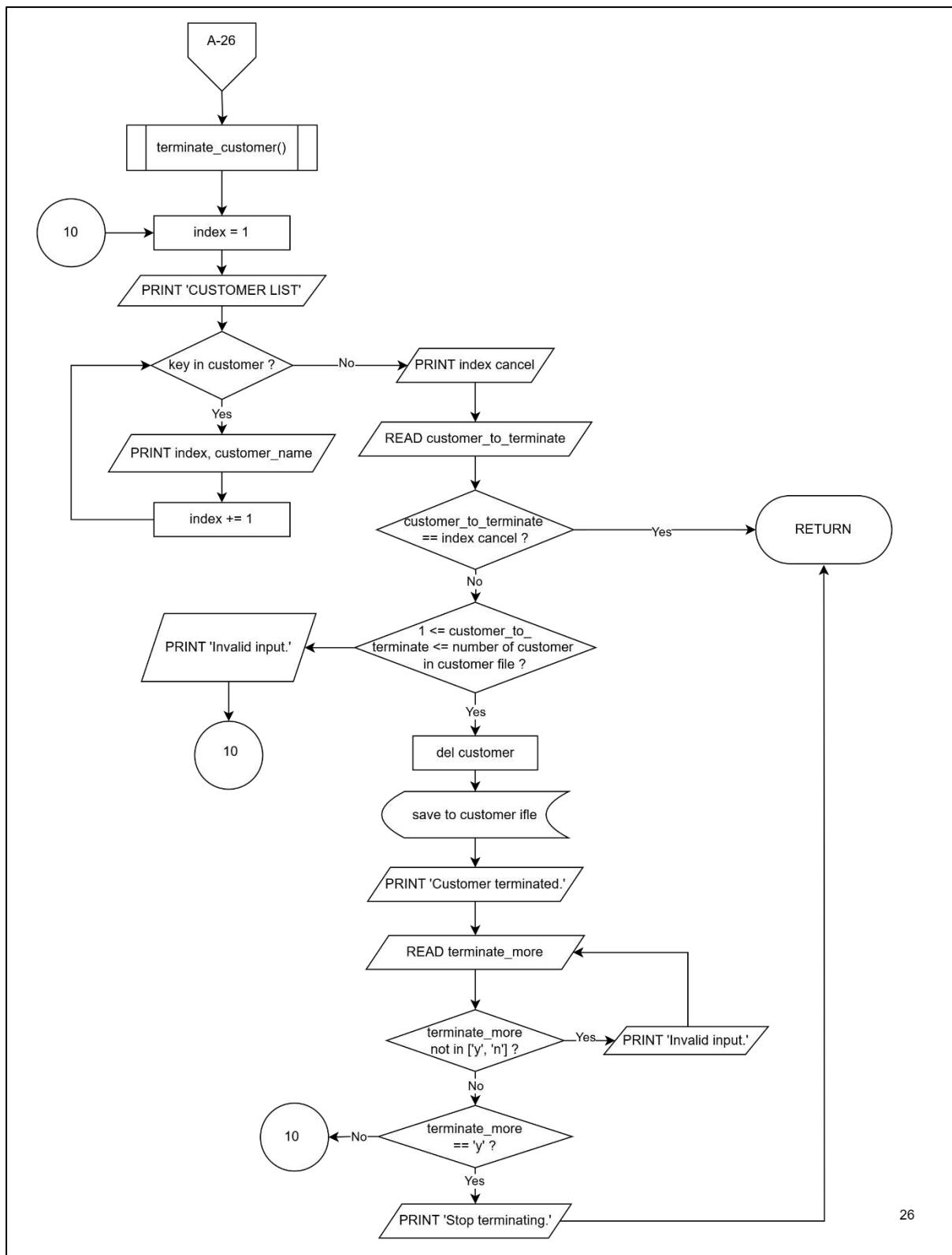
23



24

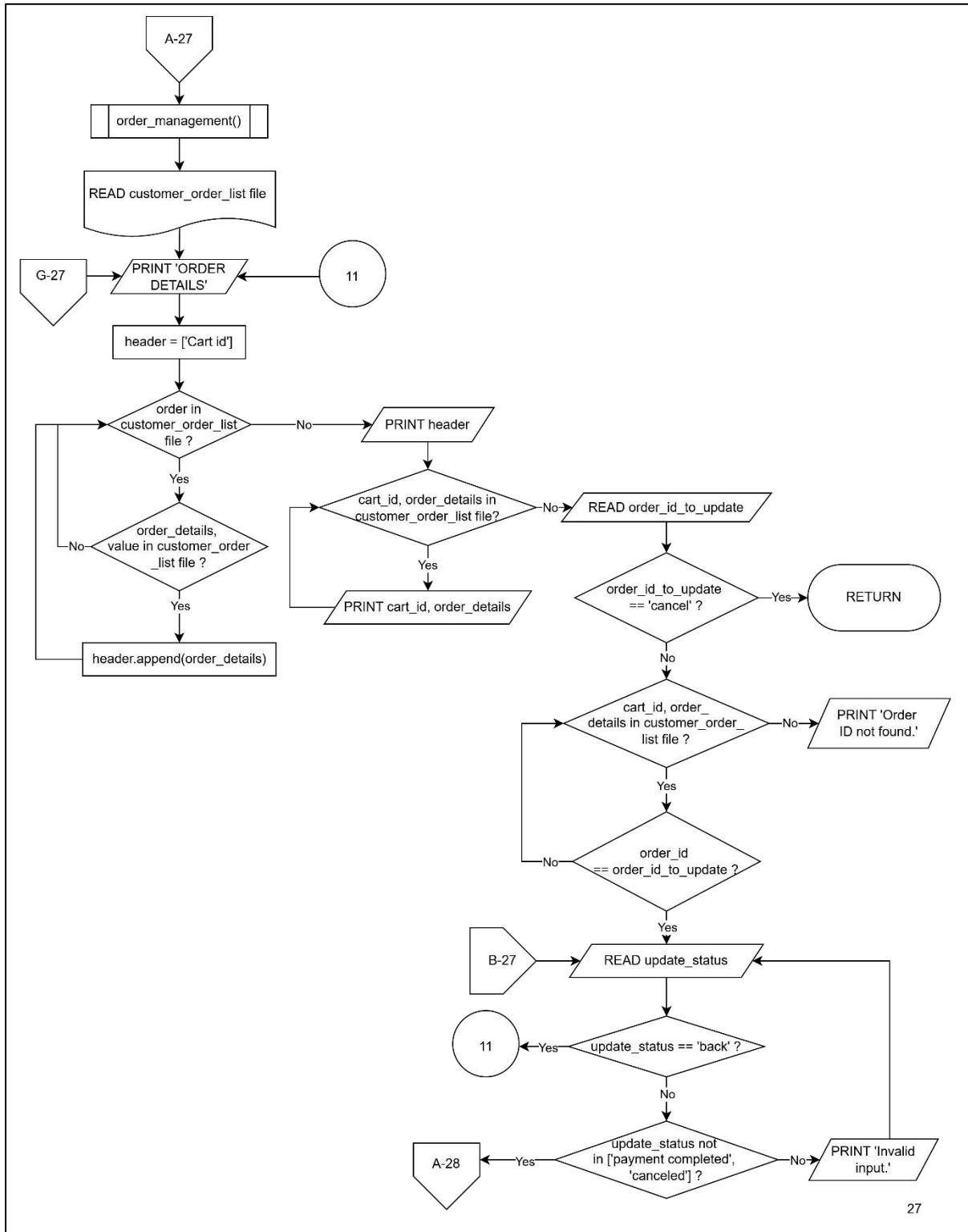


25

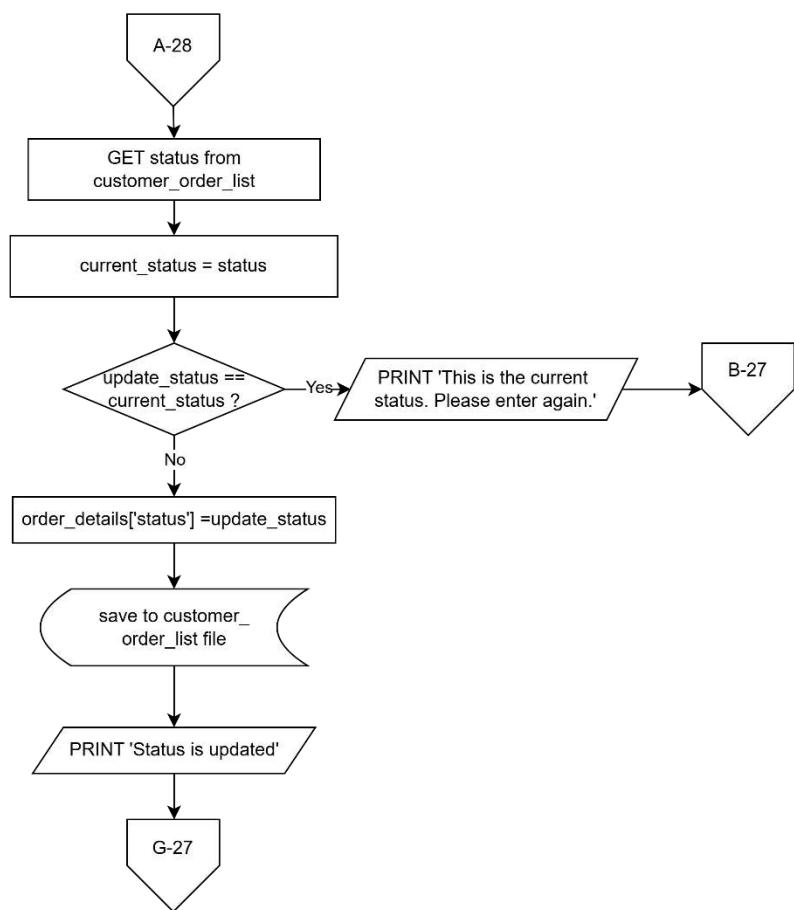


26

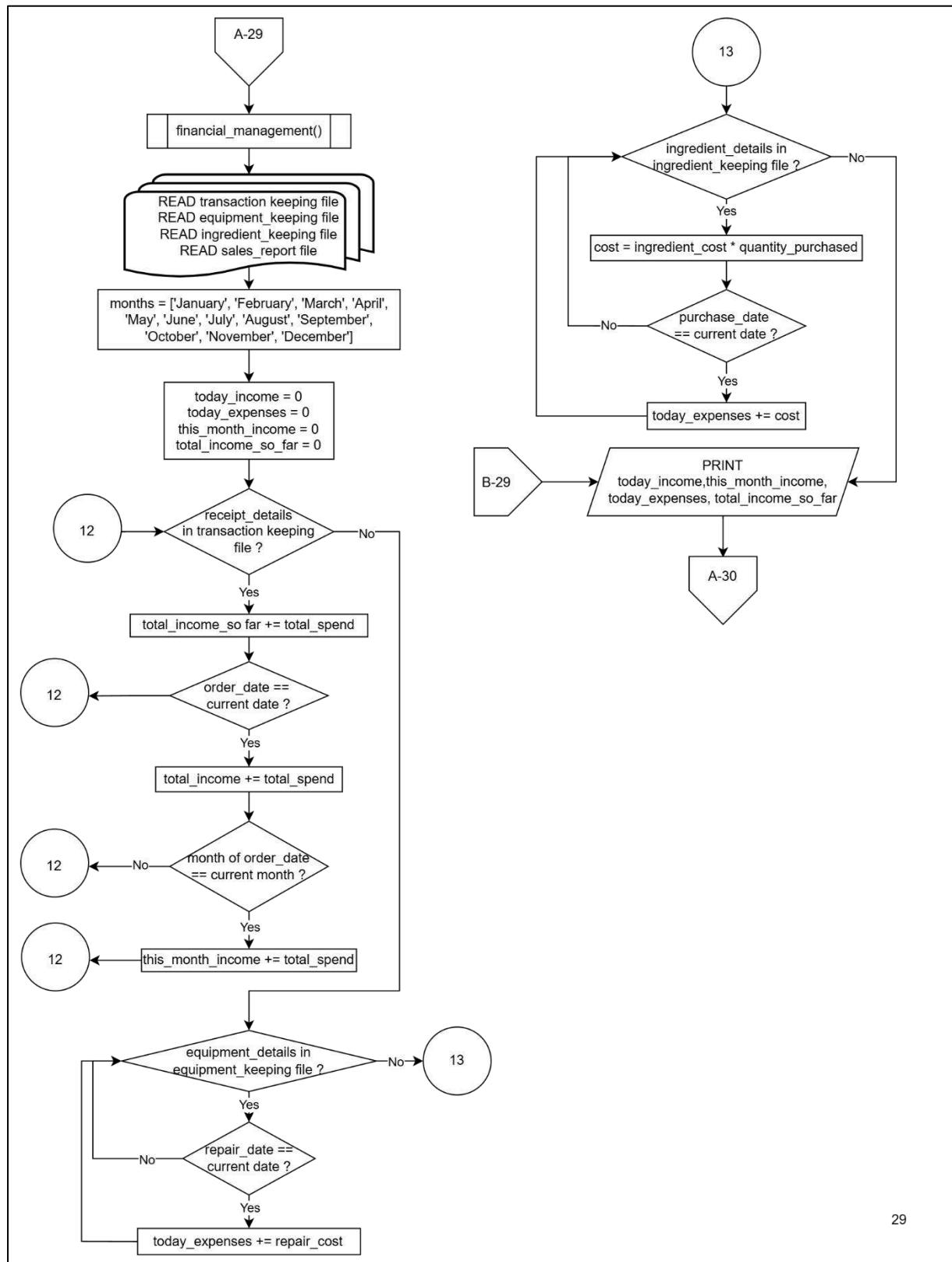
3.2.2 Order Management

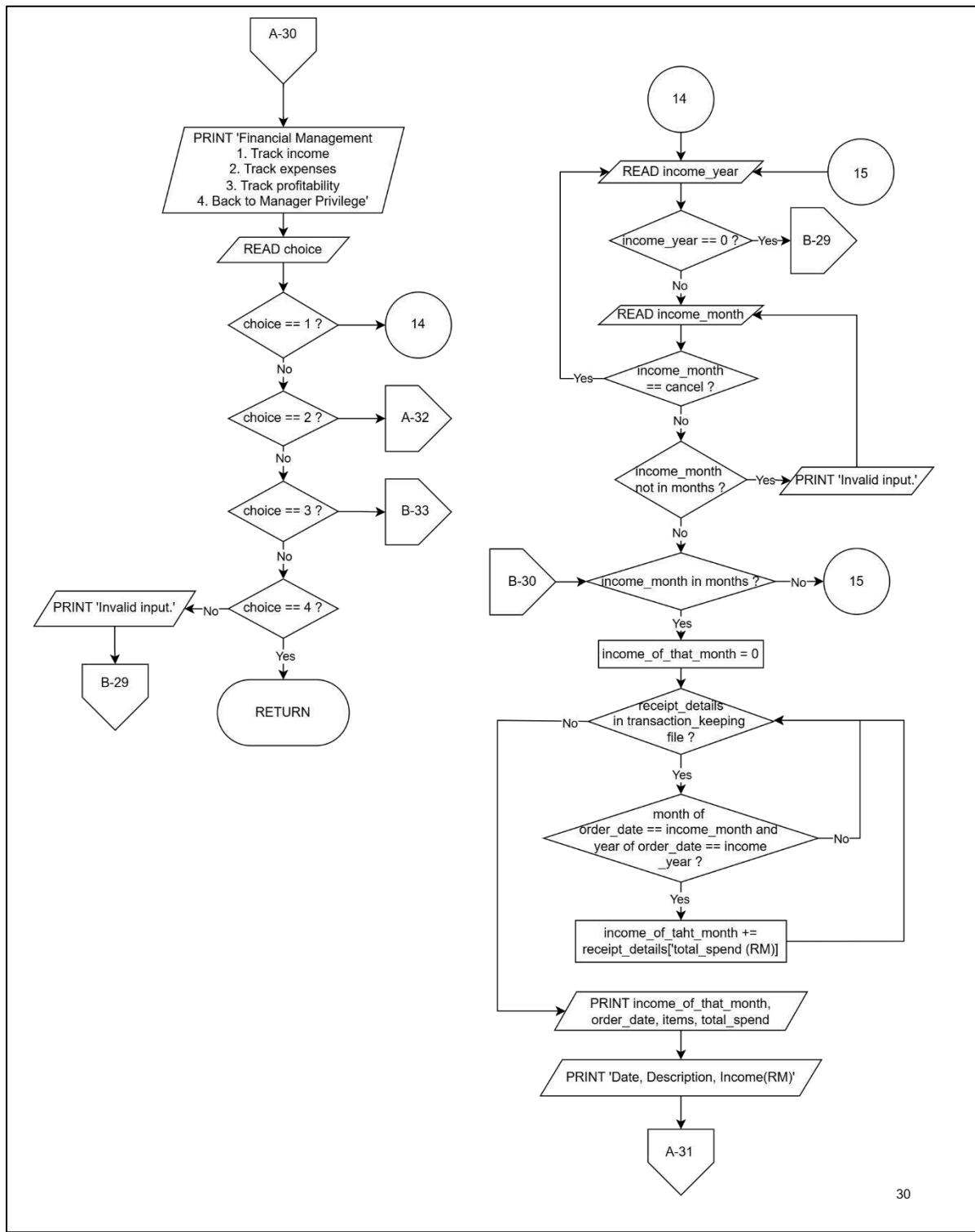


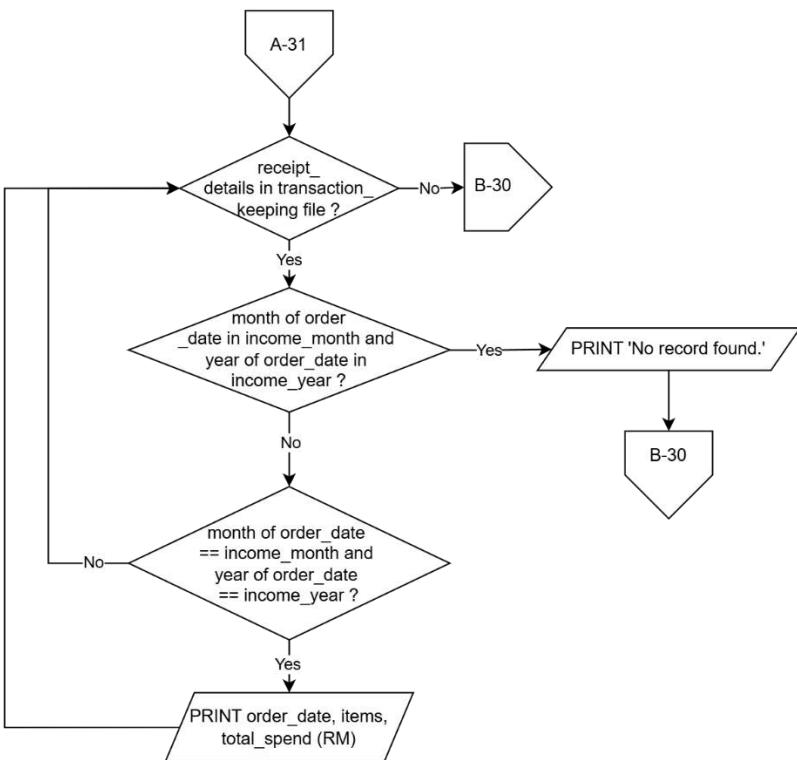
27

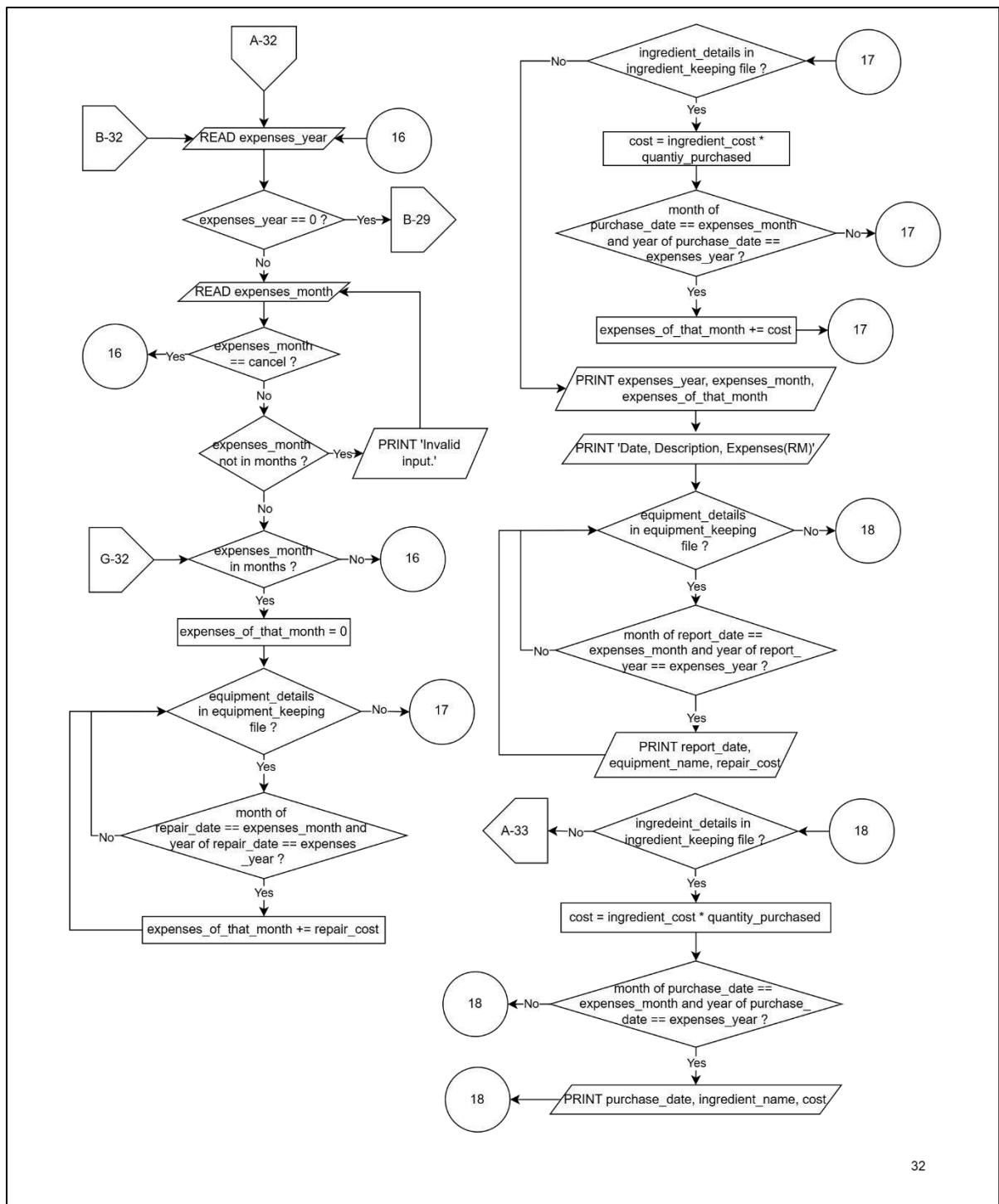


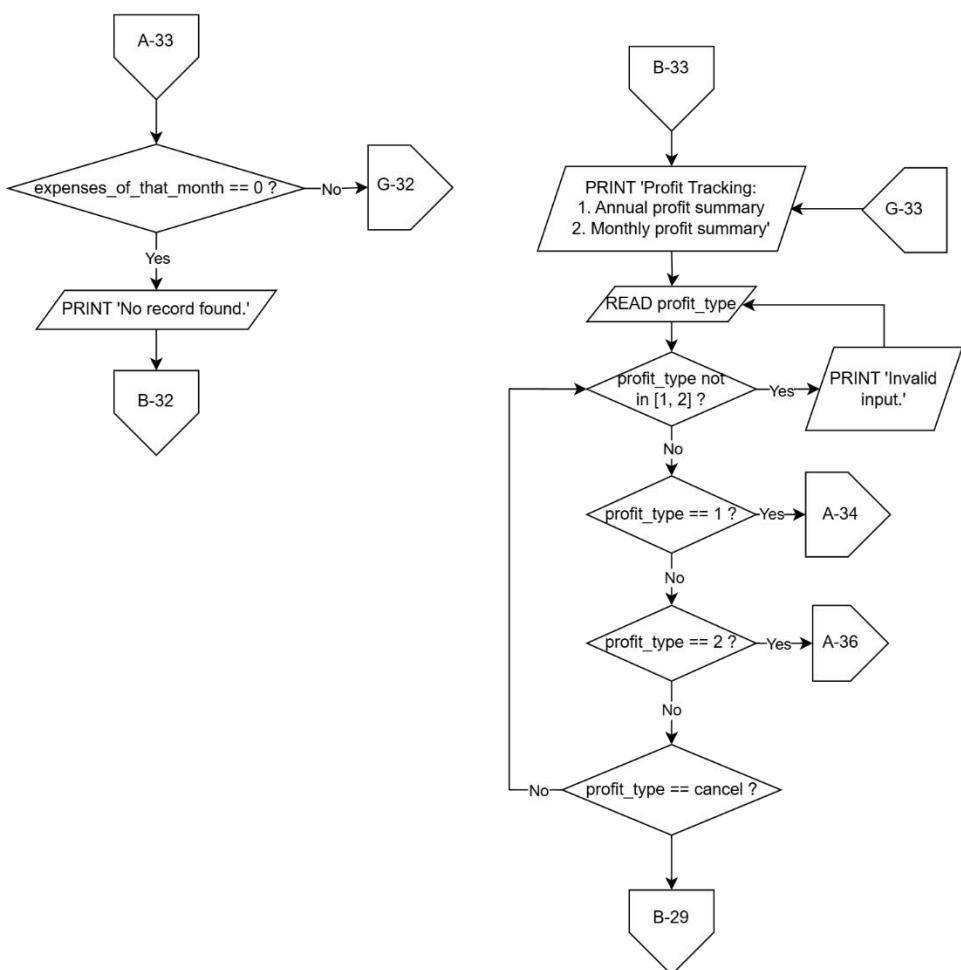
3.2.3 Financial Management

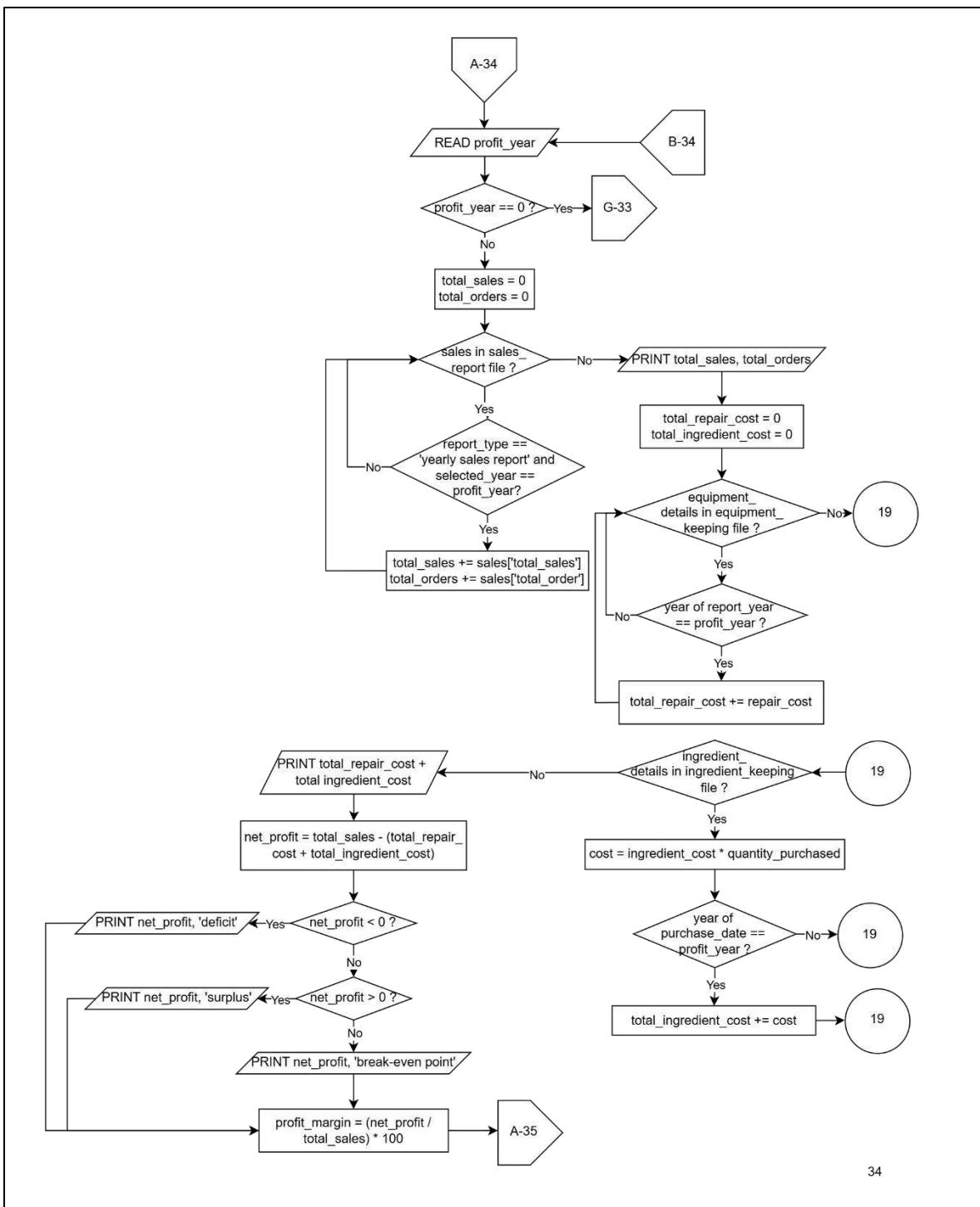




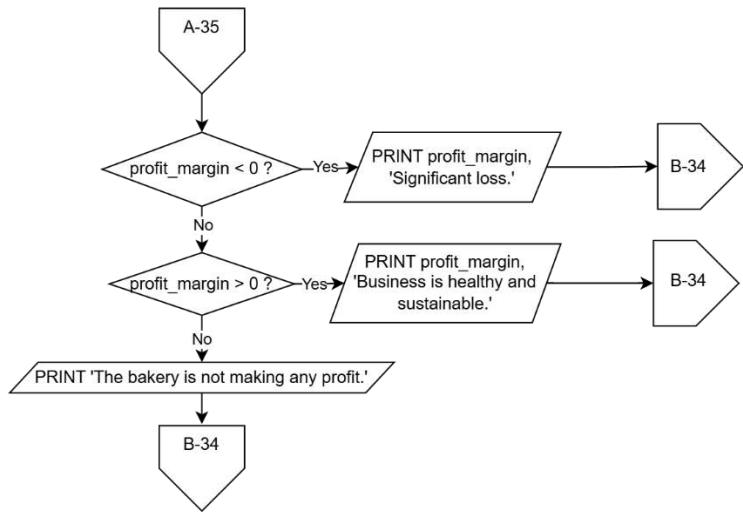


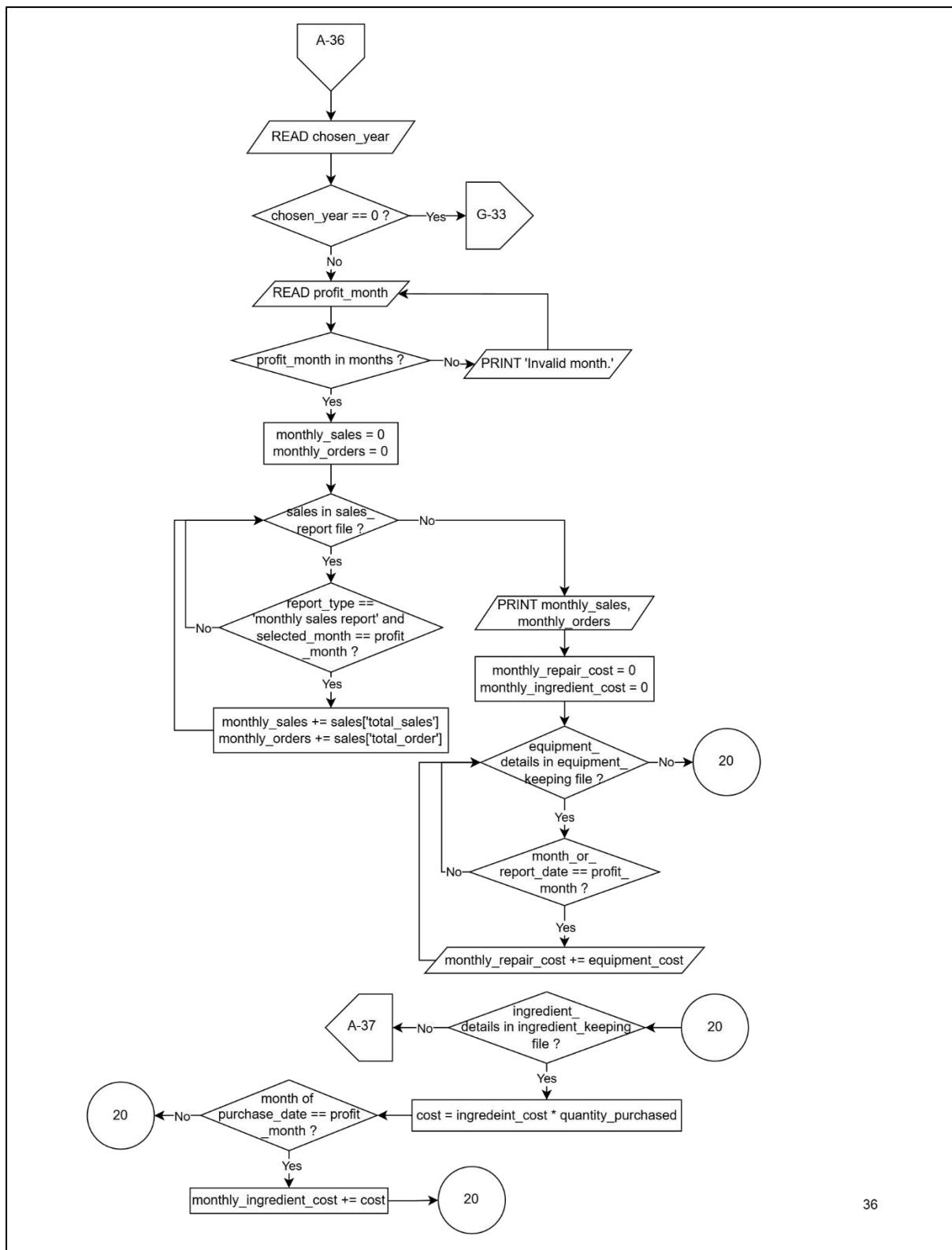




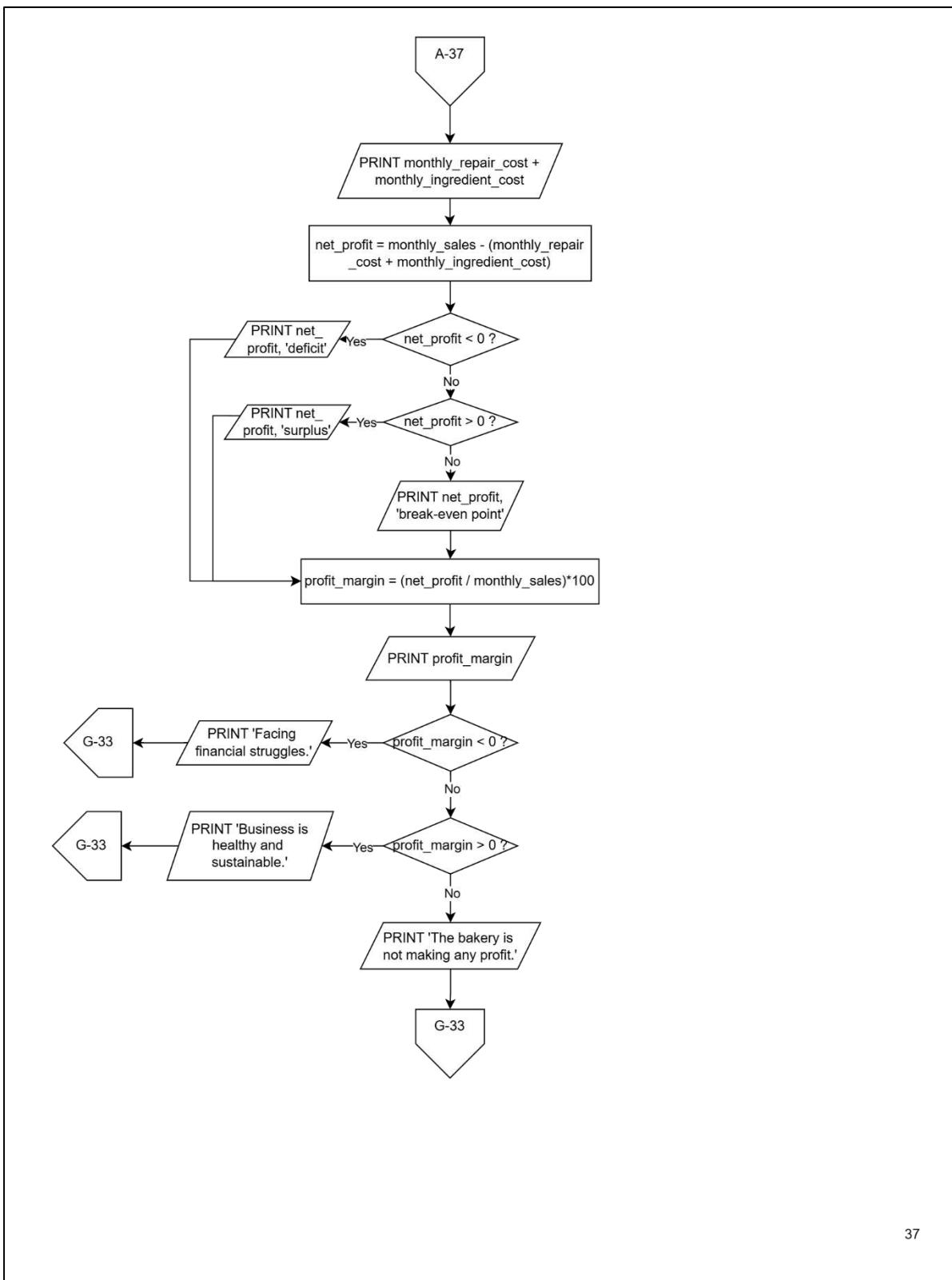


34

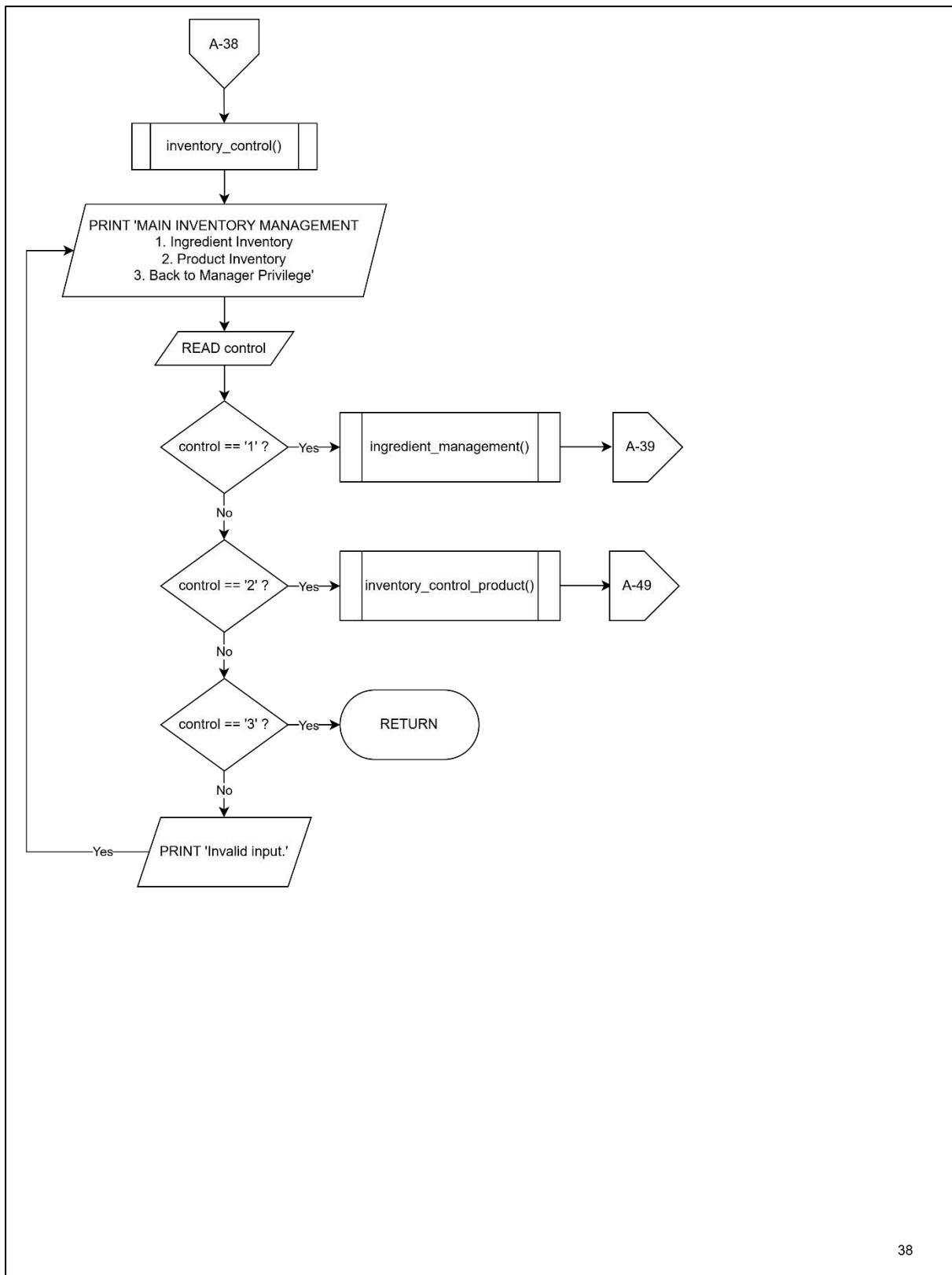




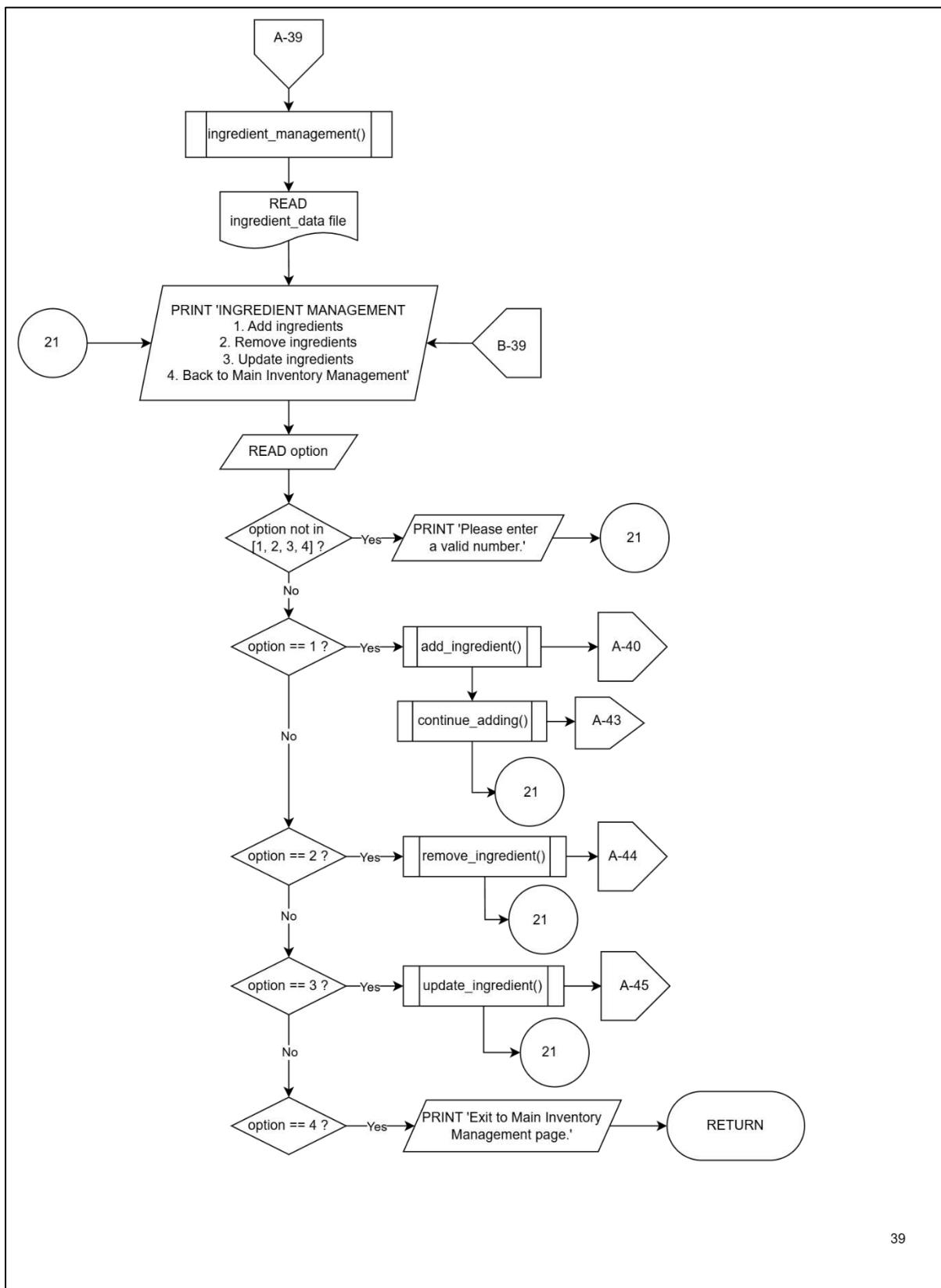
36

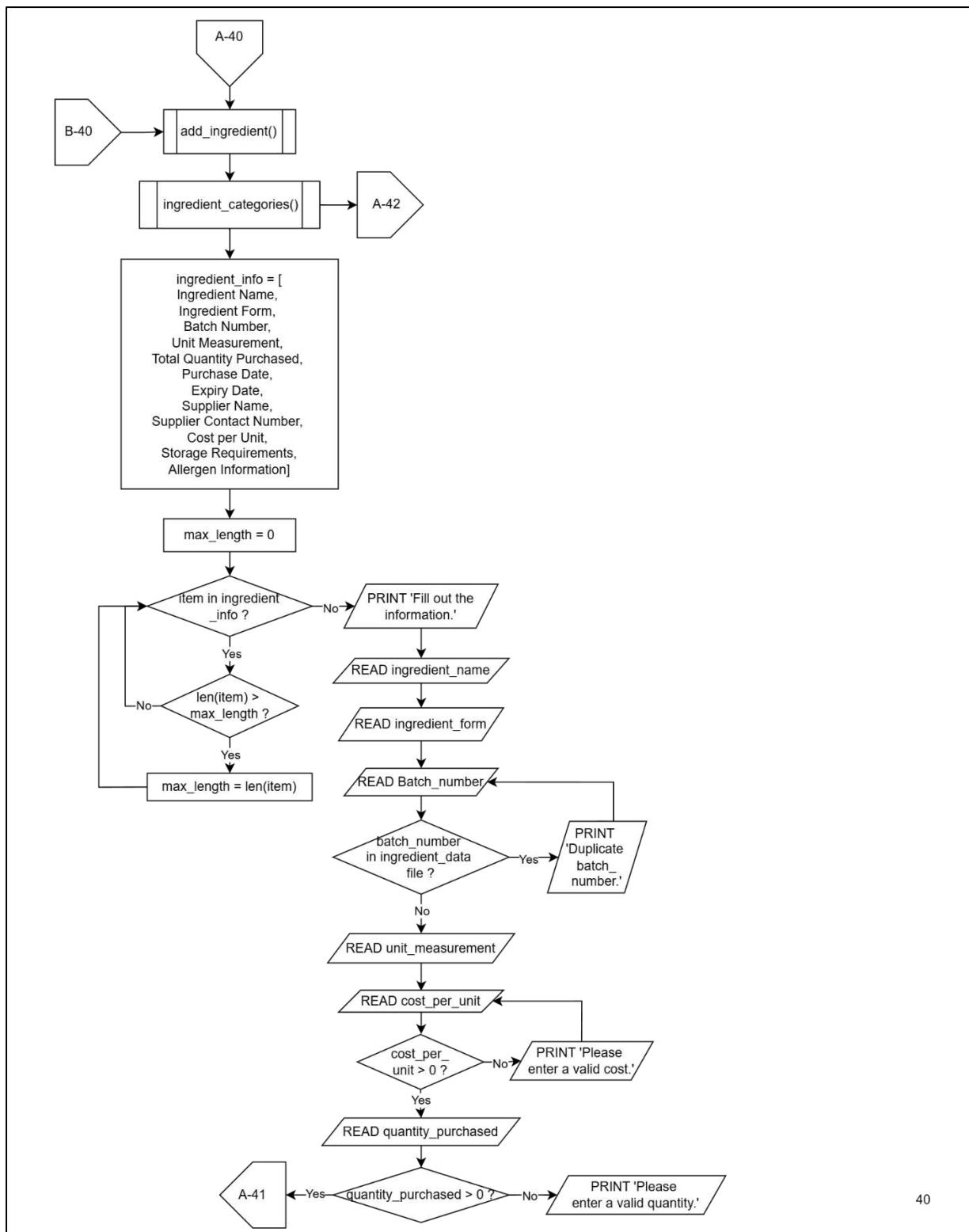


3.2.4 Inventory Control

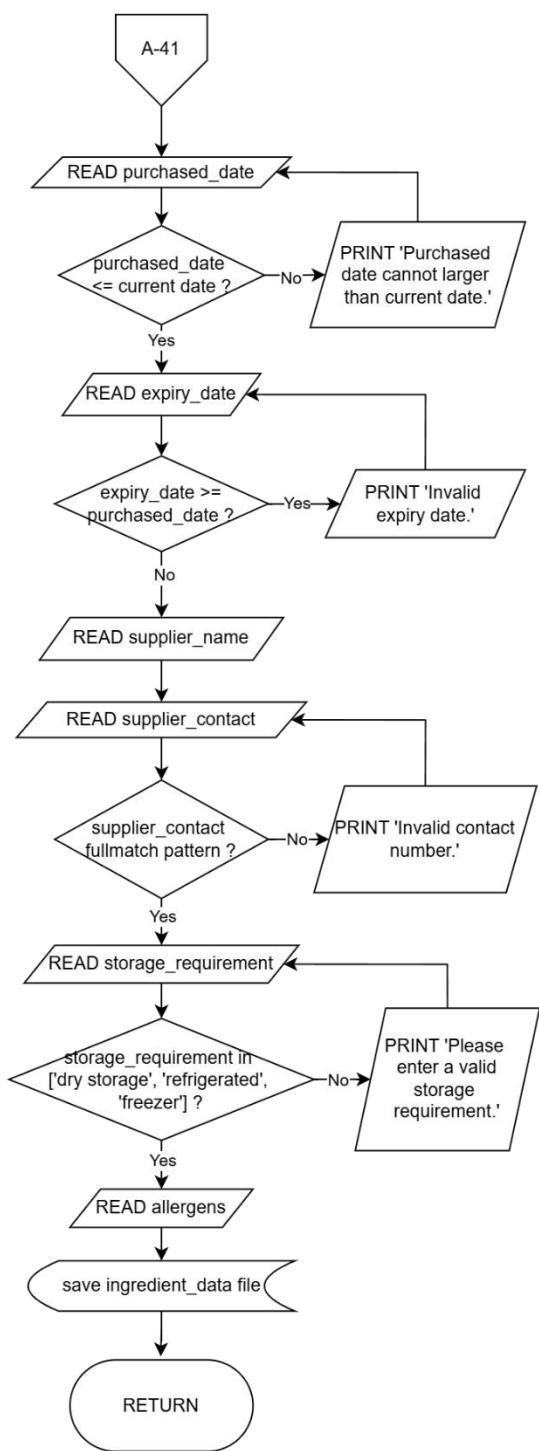


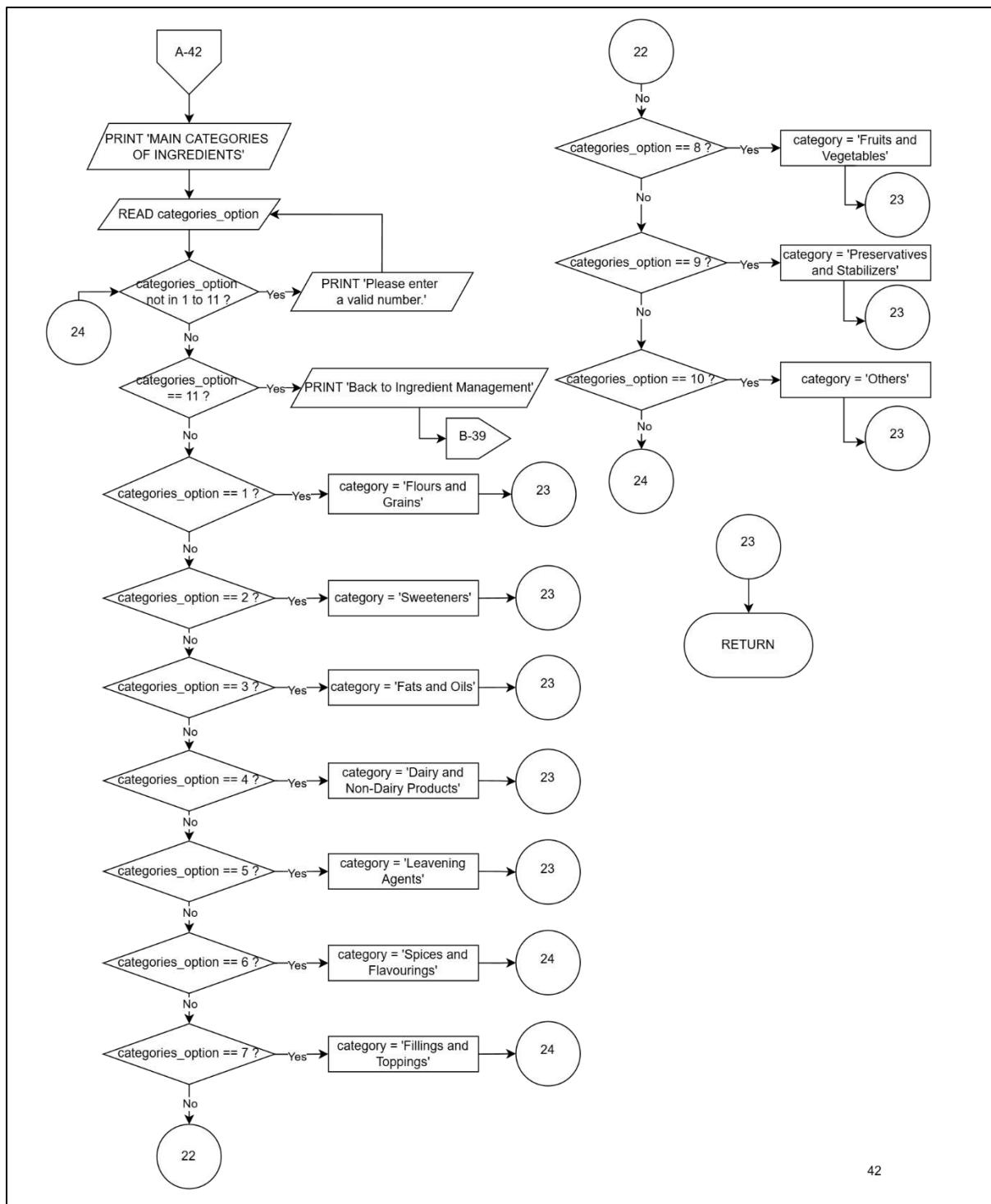
Ingredient inventory



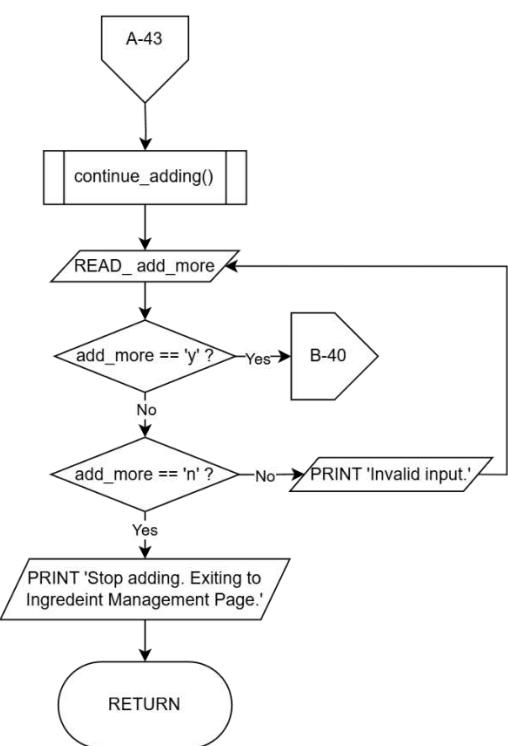


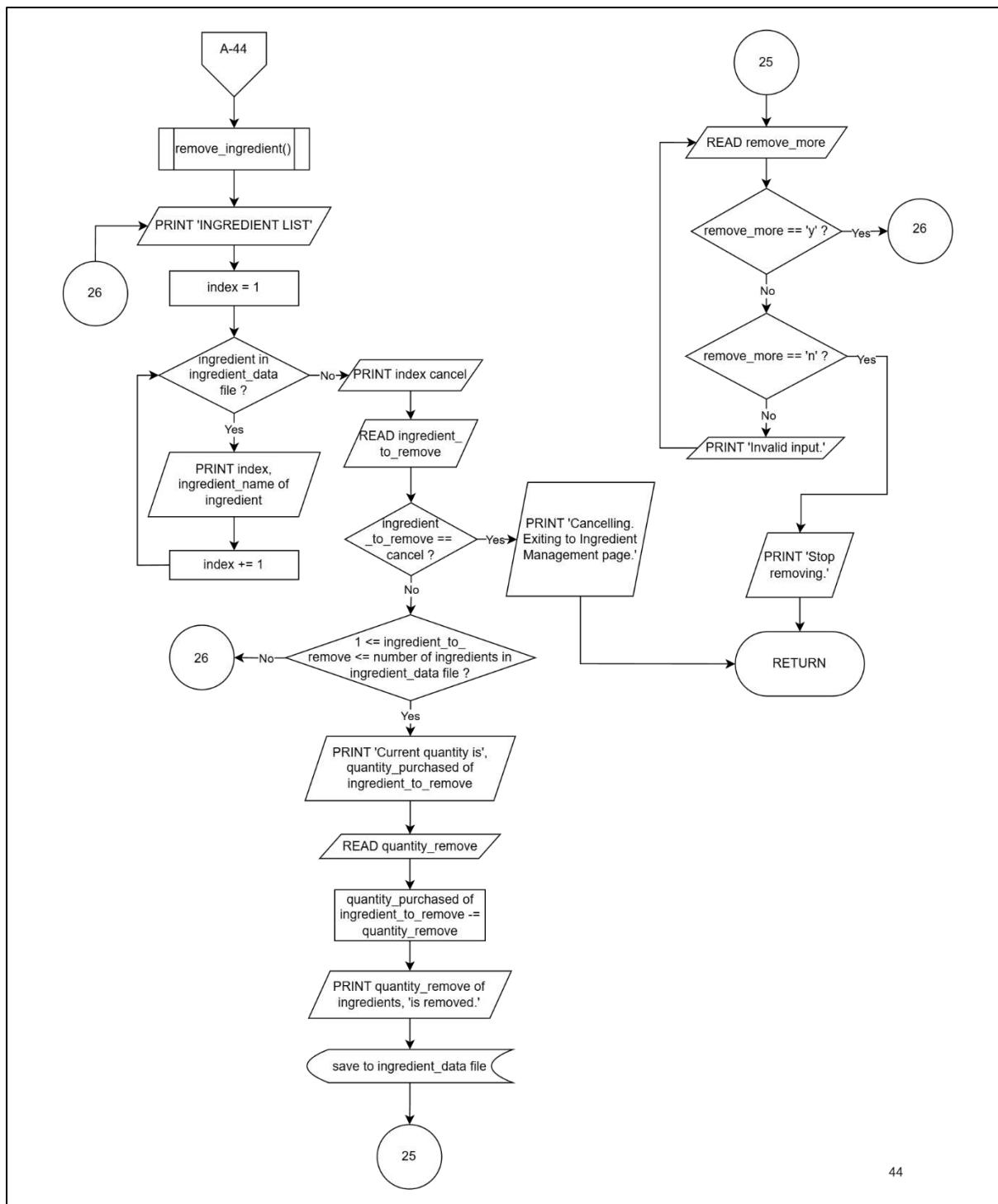
40

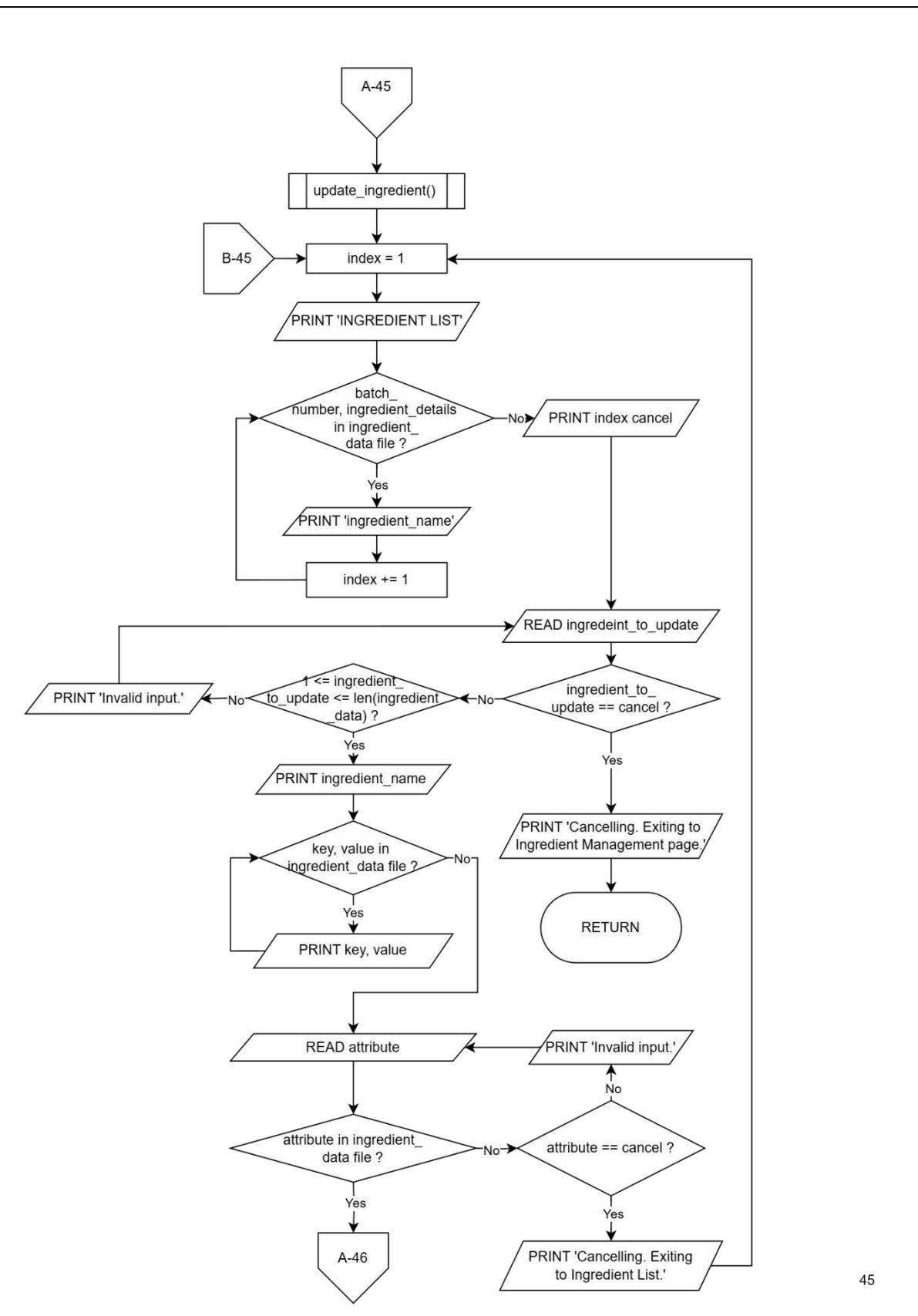




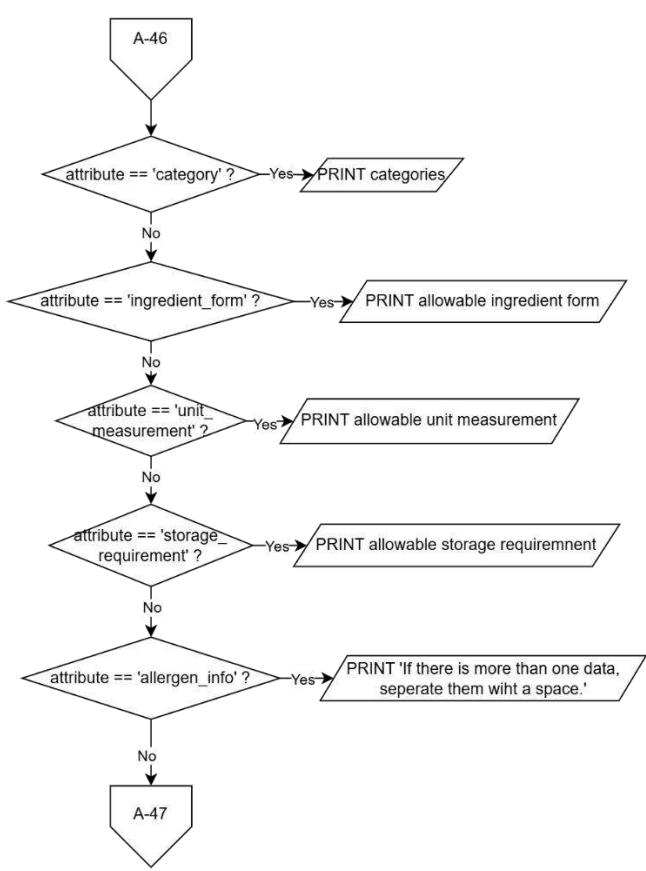
42

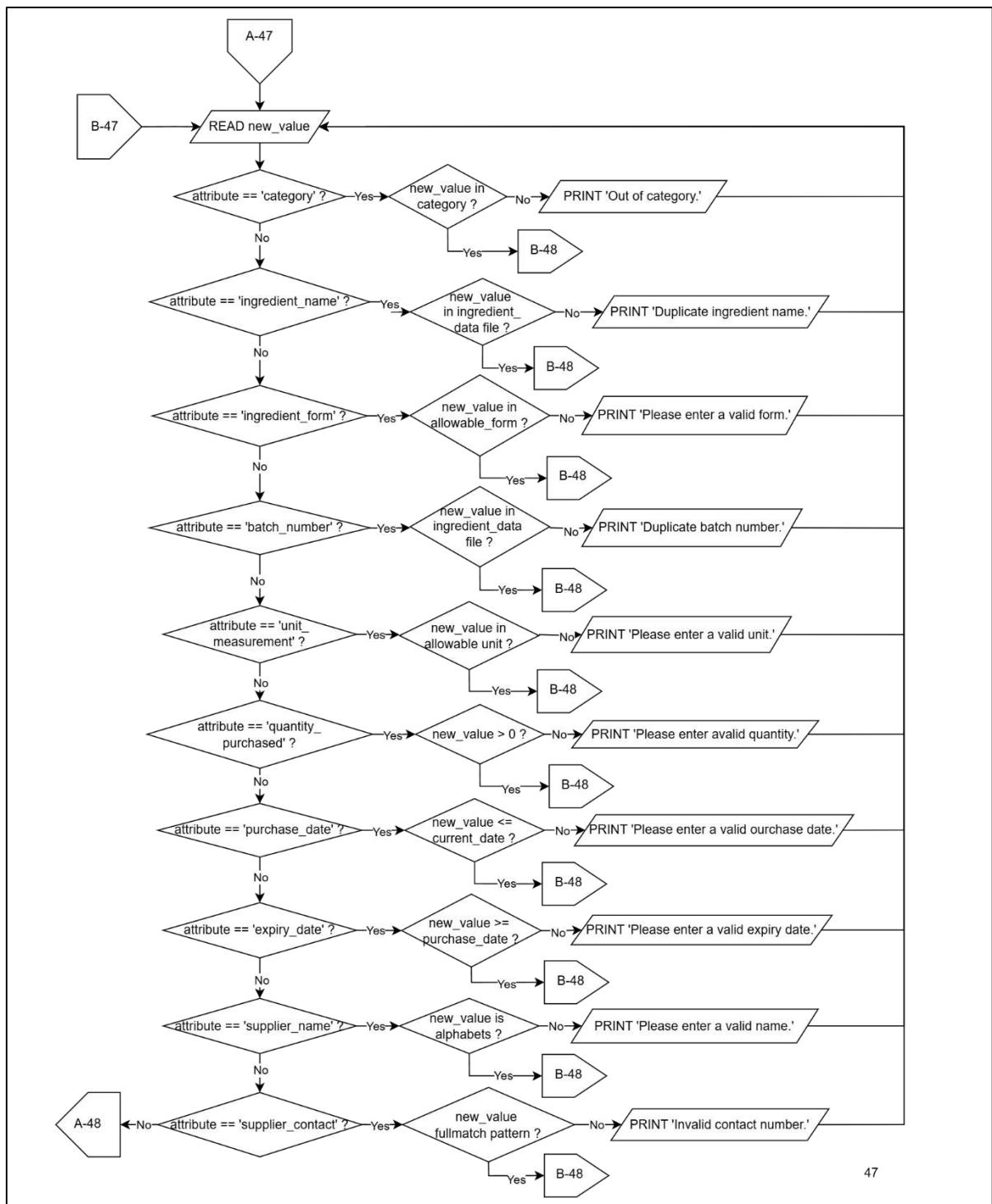




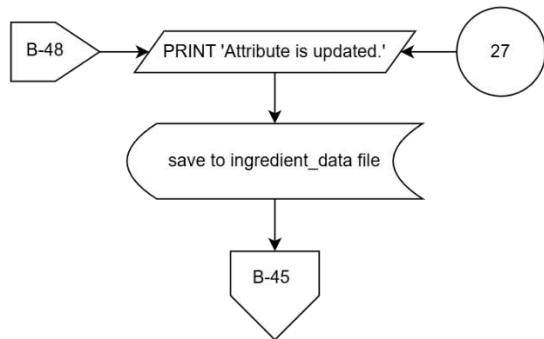
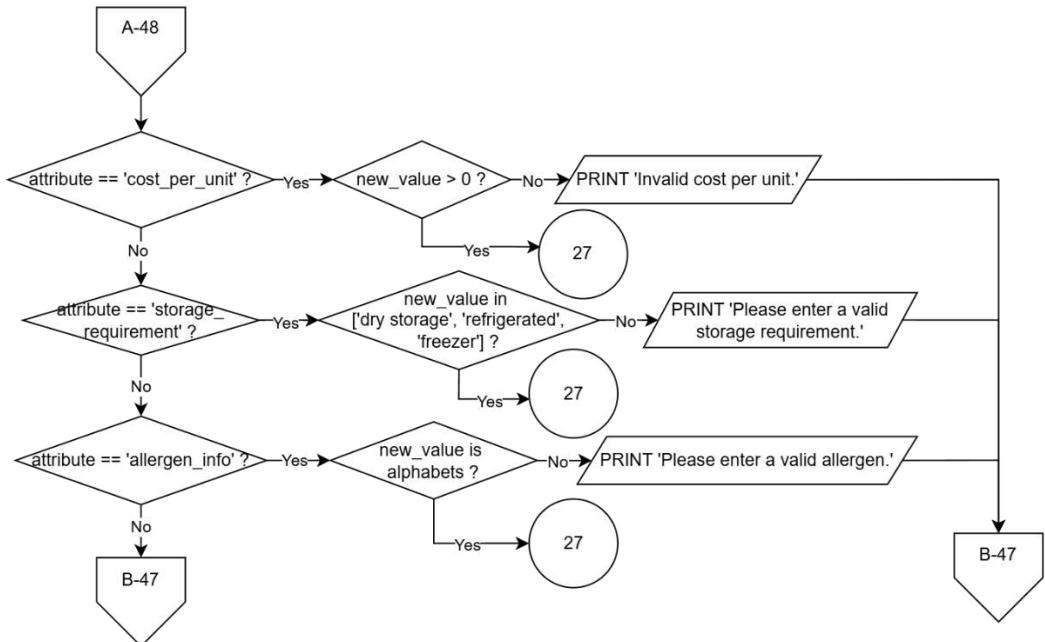


45

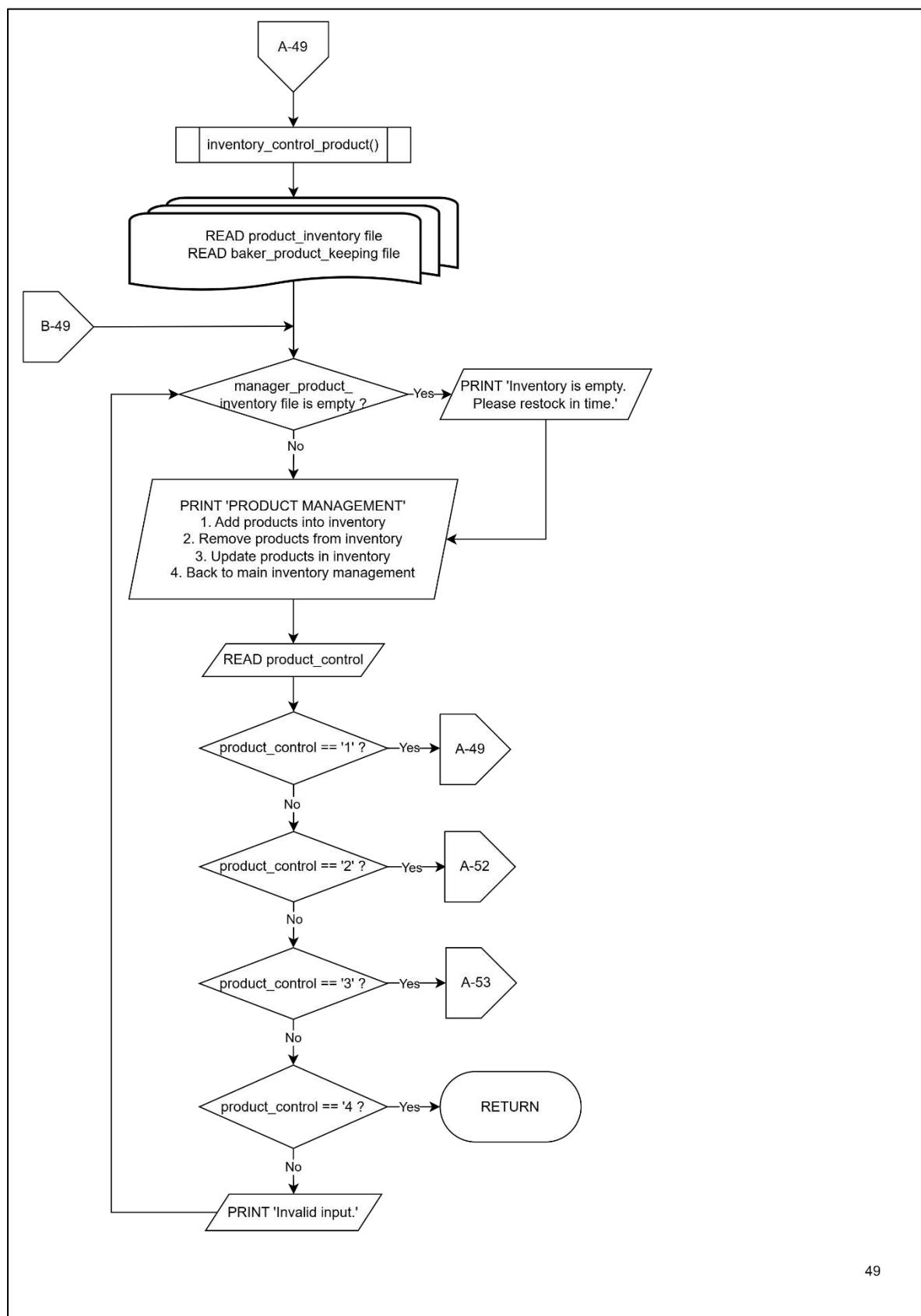


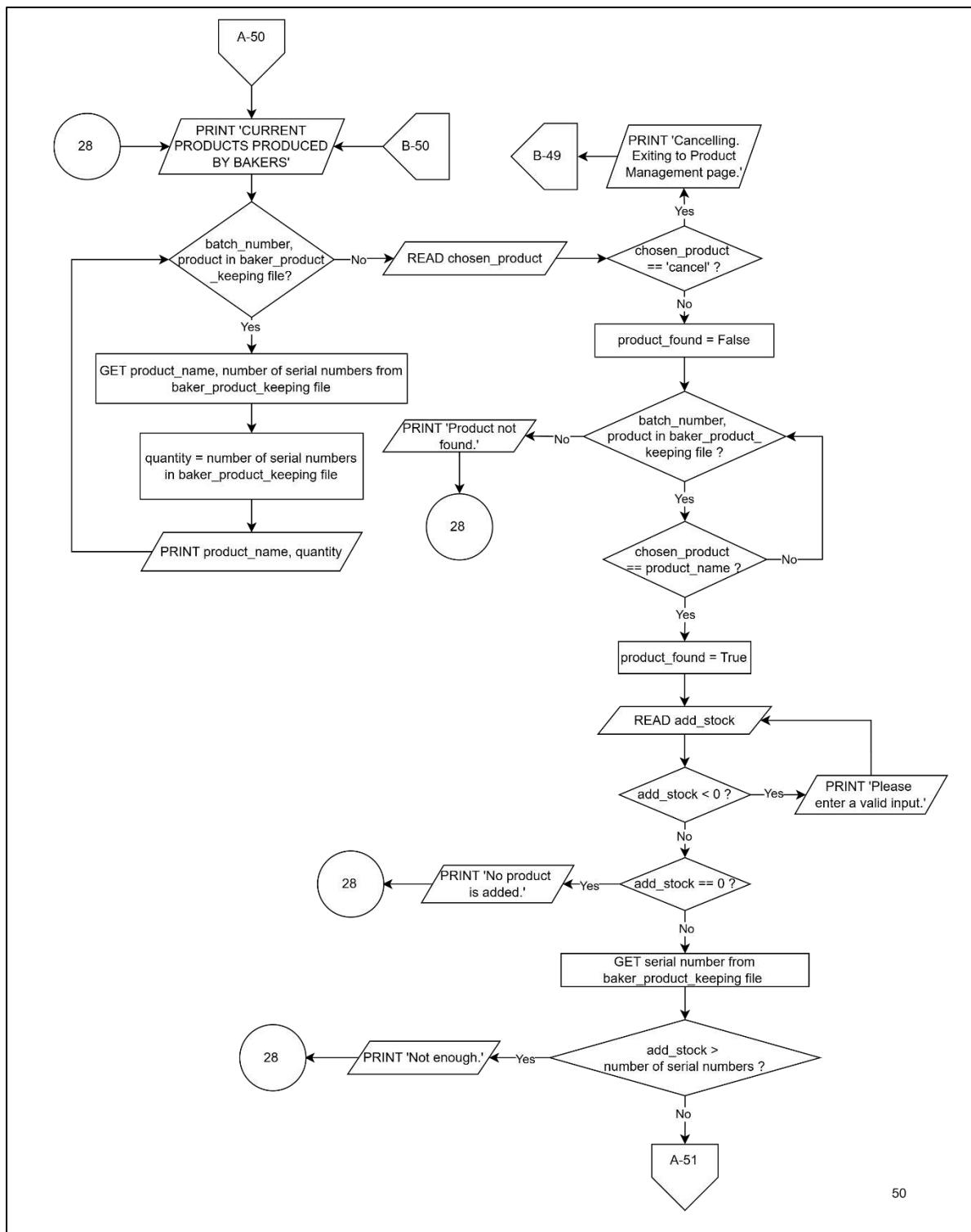


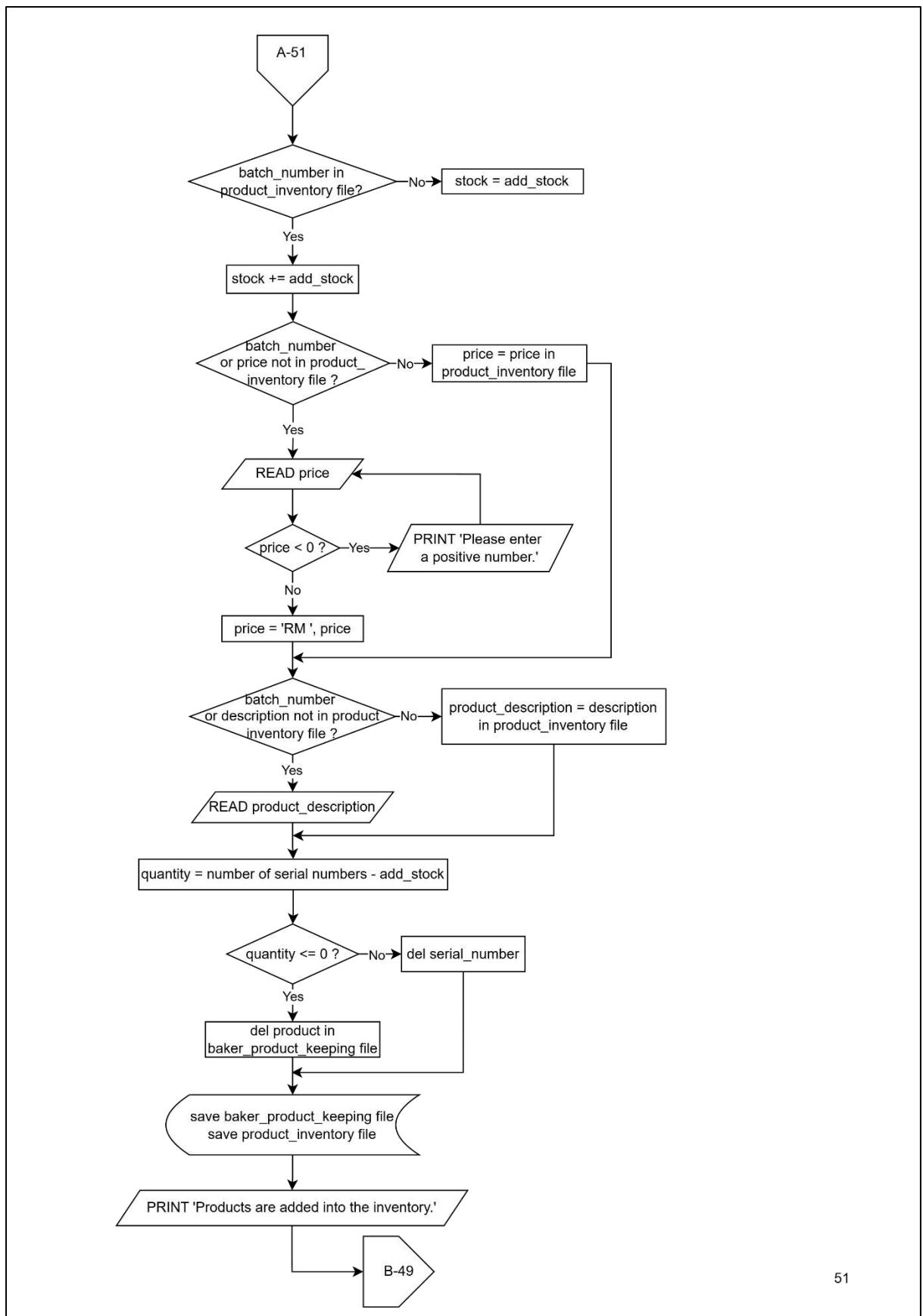
47

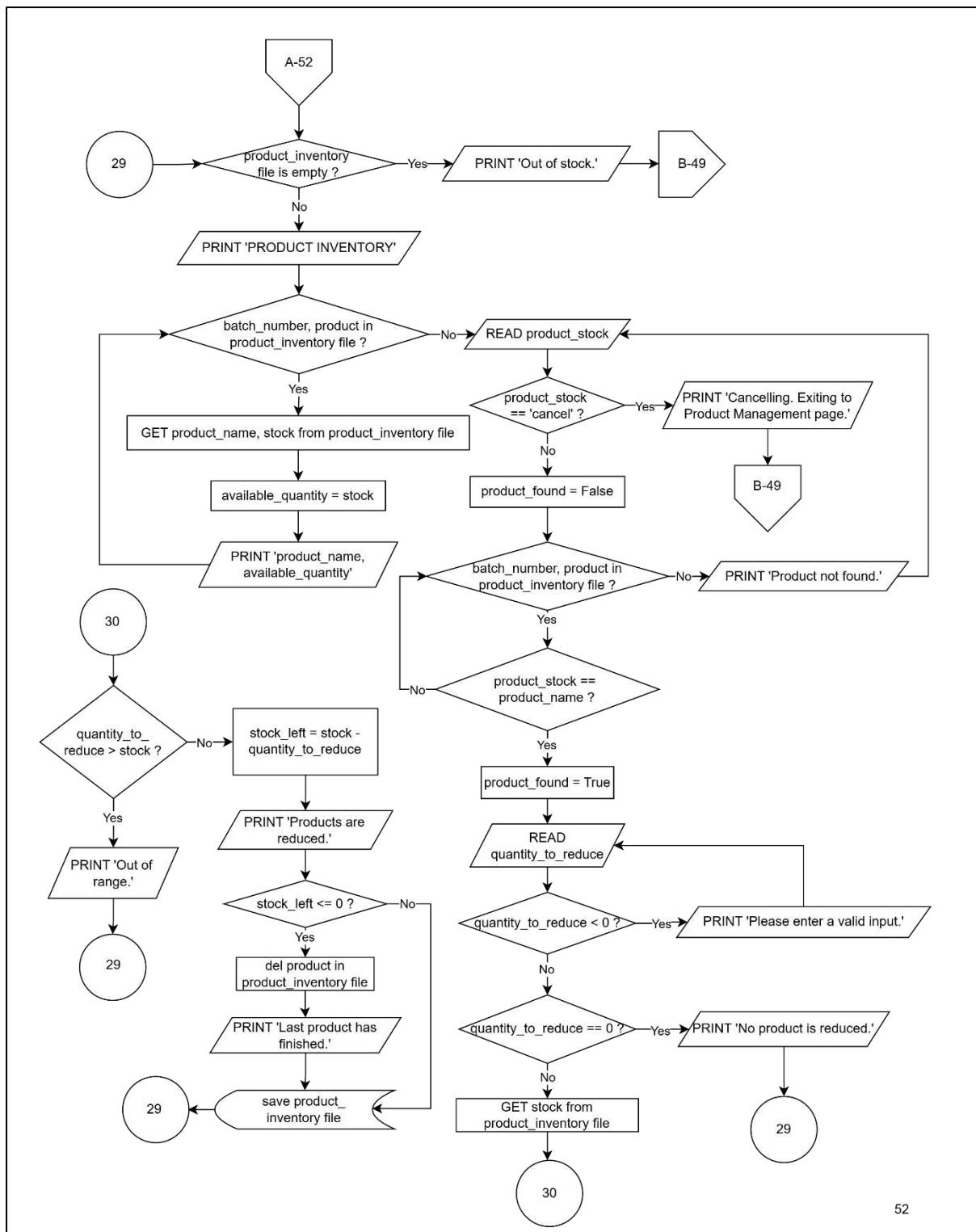


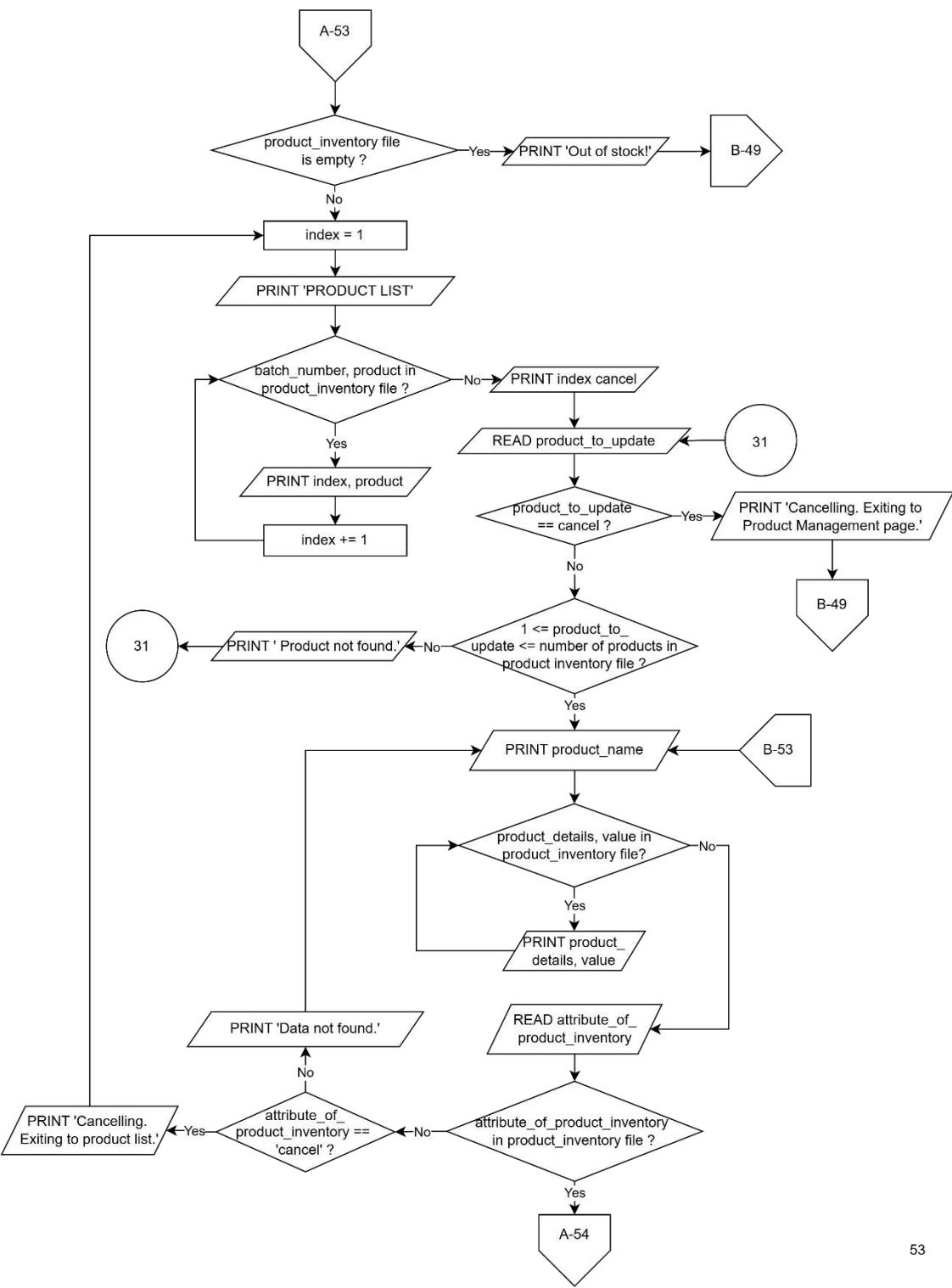
Product inventory



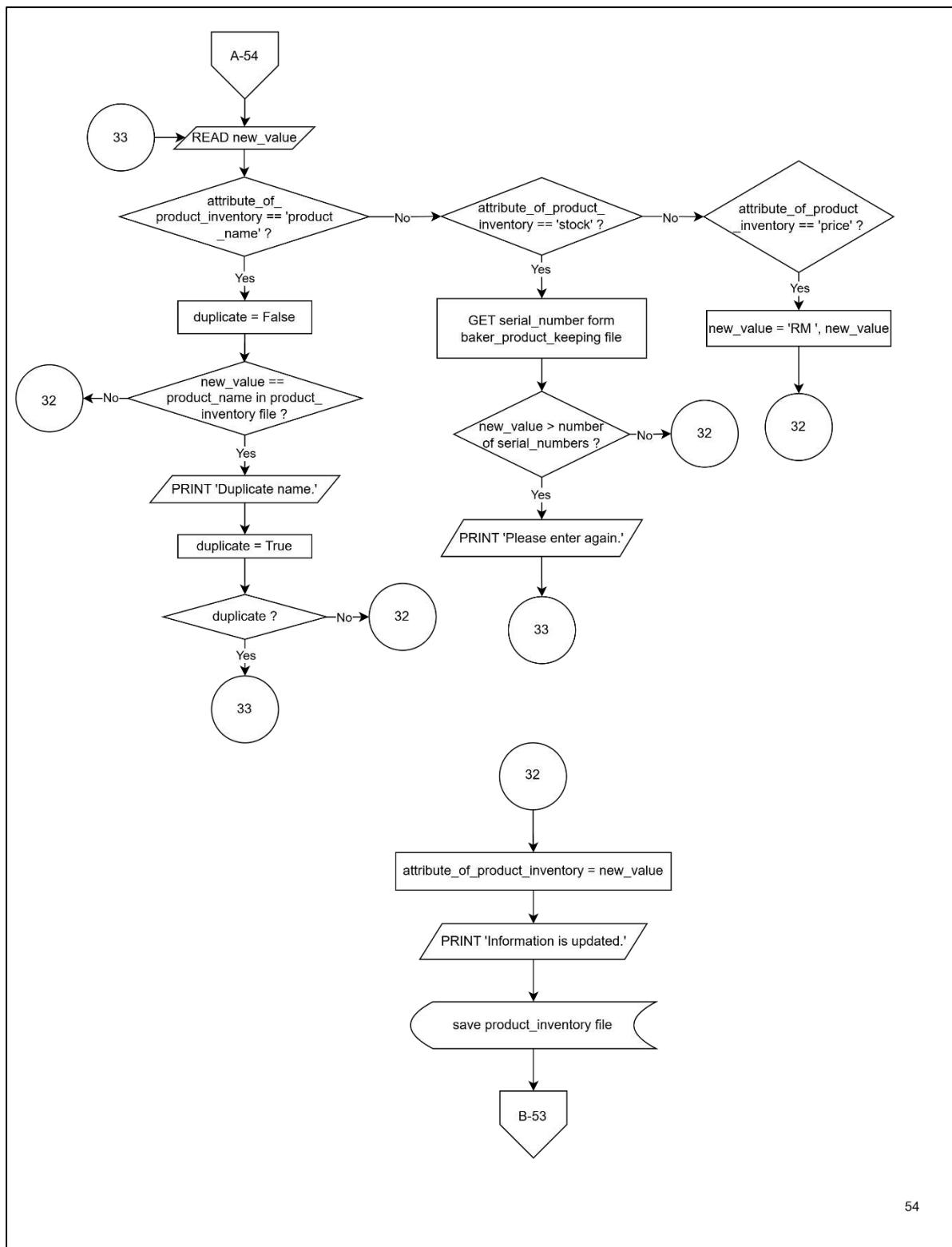




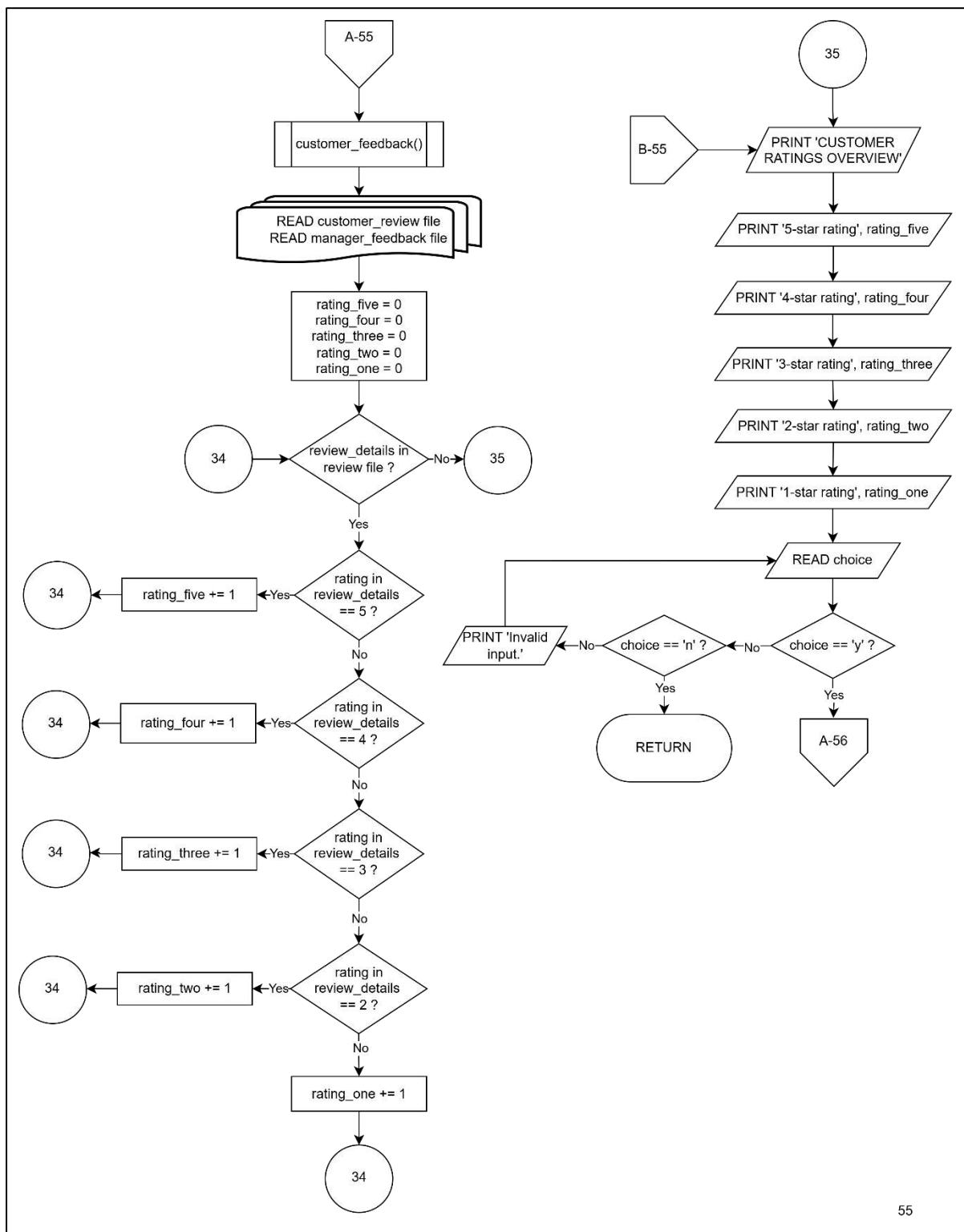




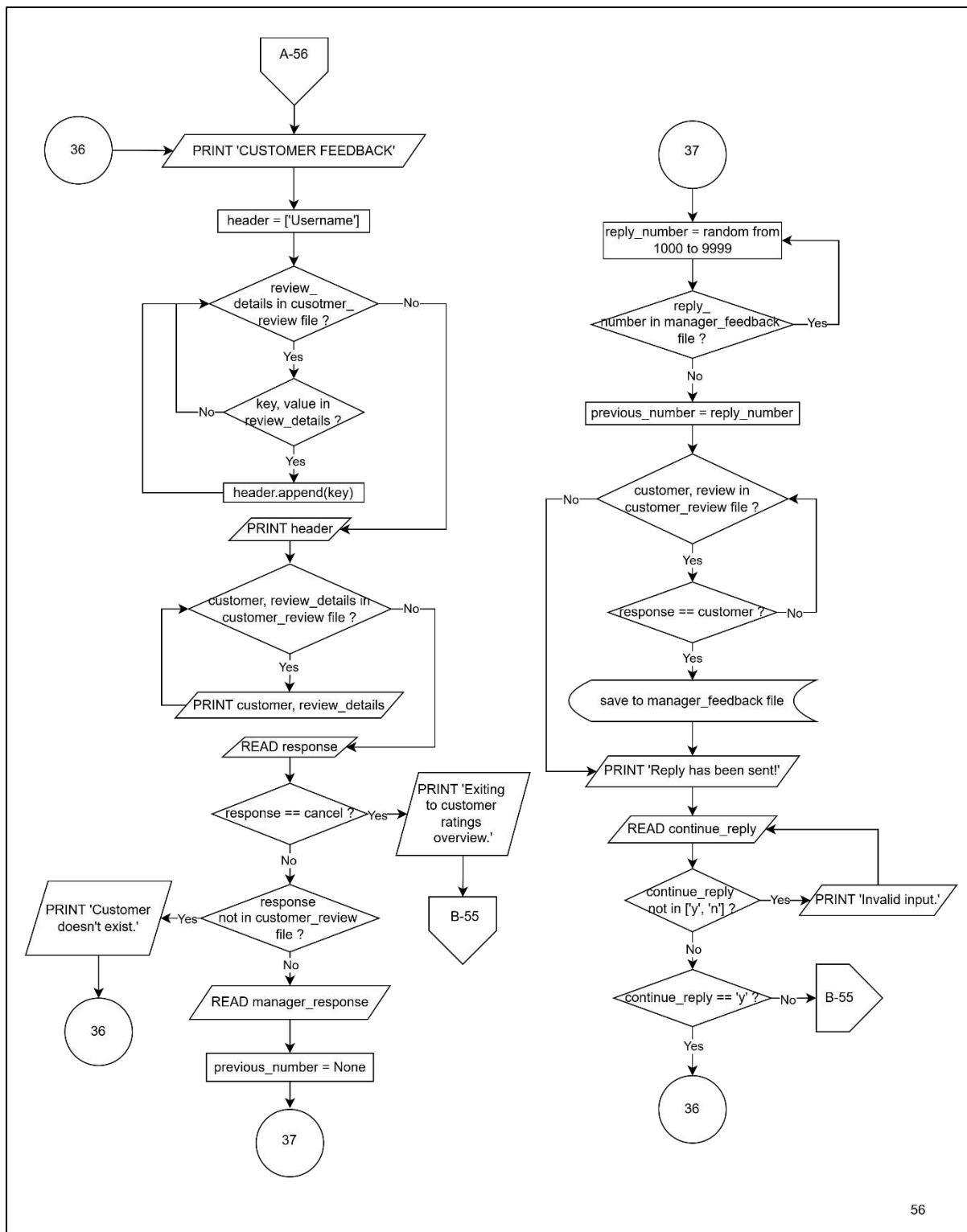
53



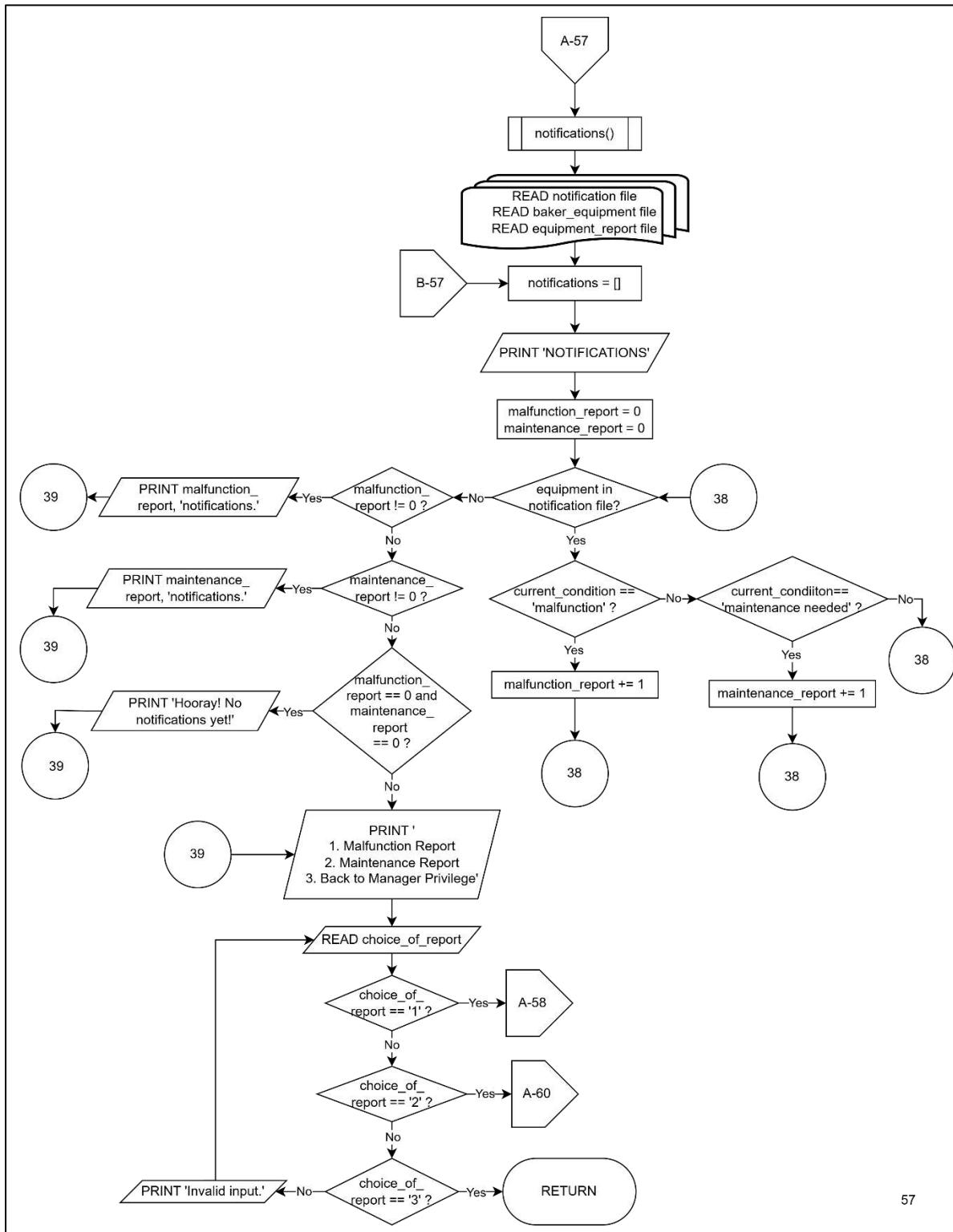
3.2.5 Customer Feedback



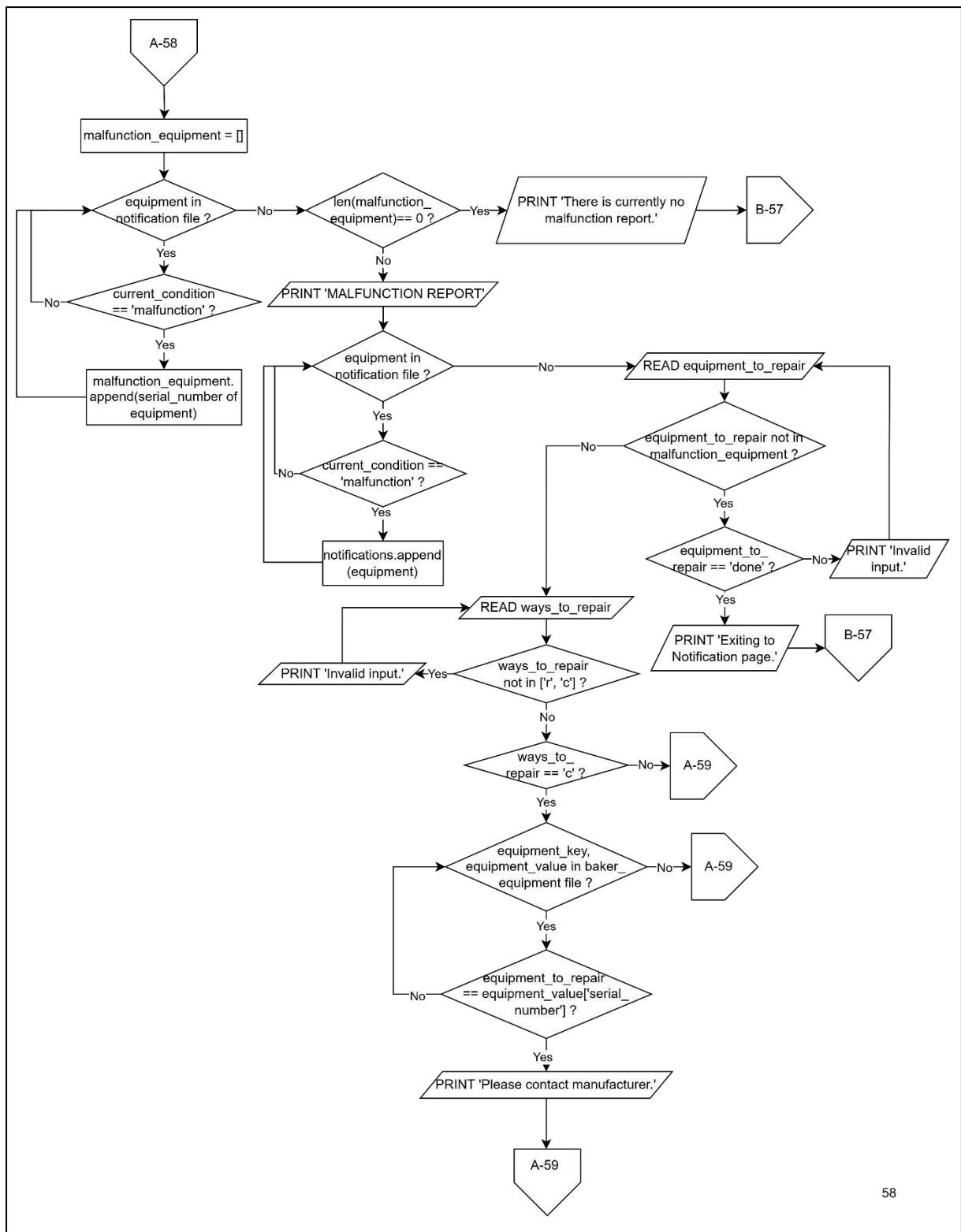
55



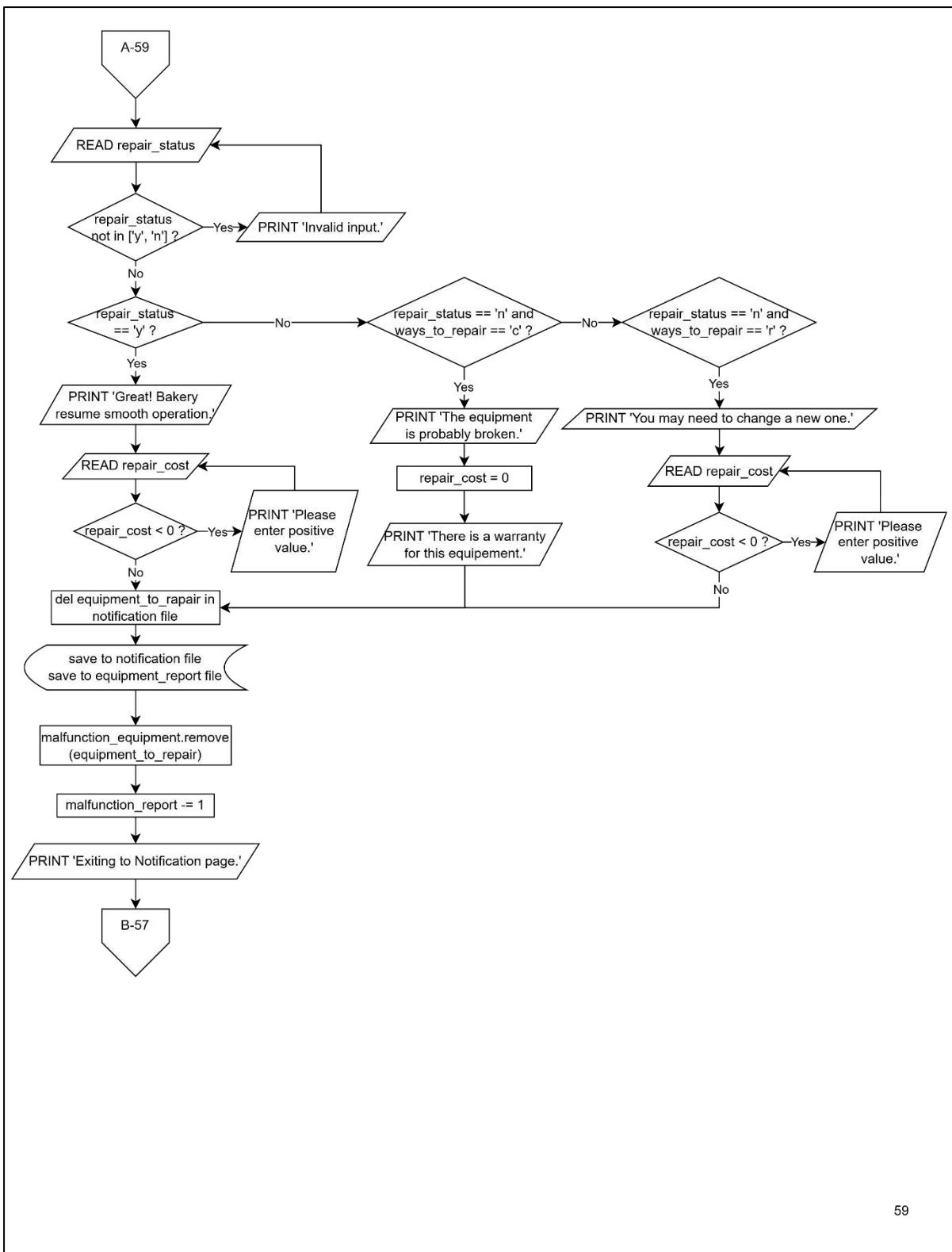
3.2.6 Notifications

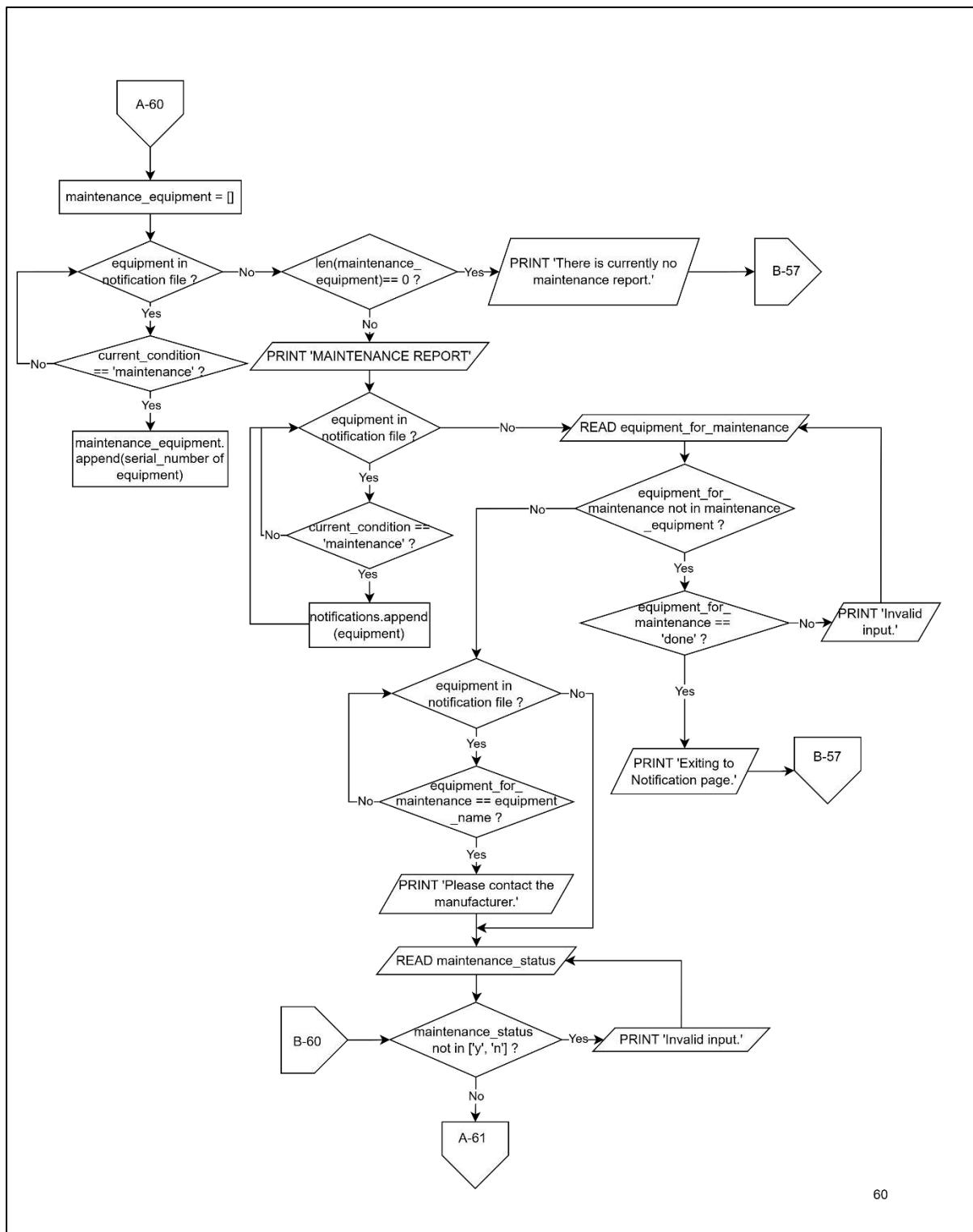


57

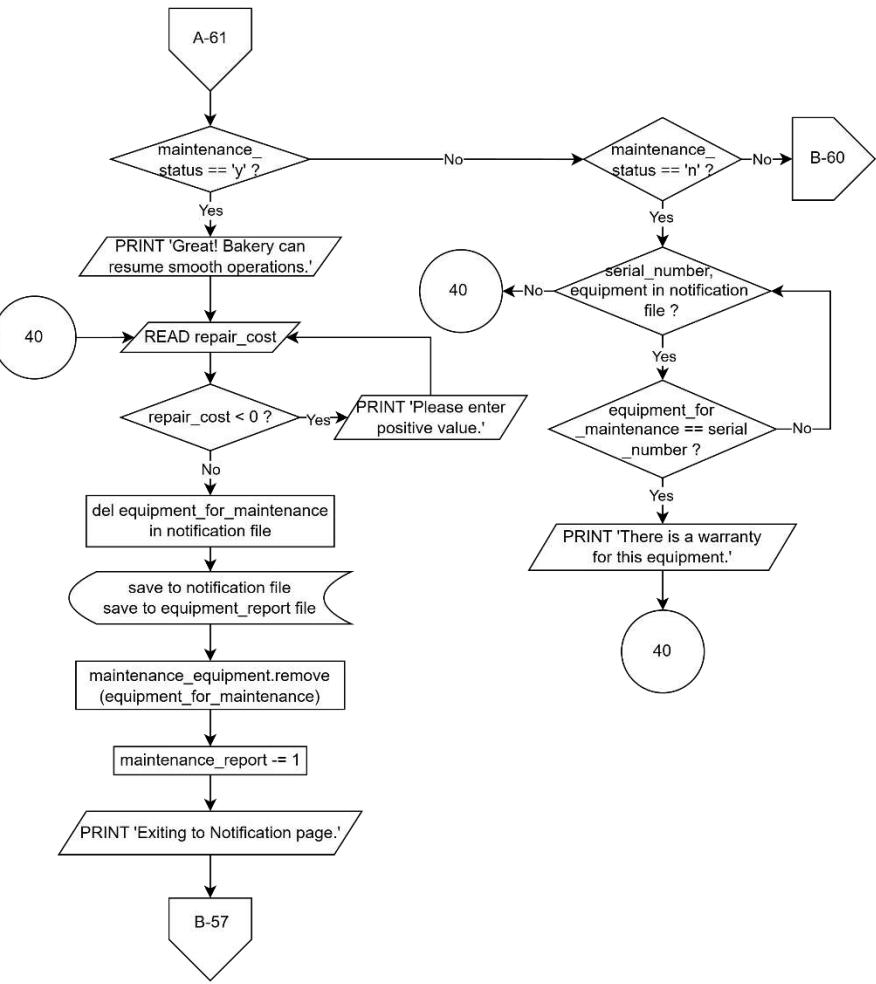


58





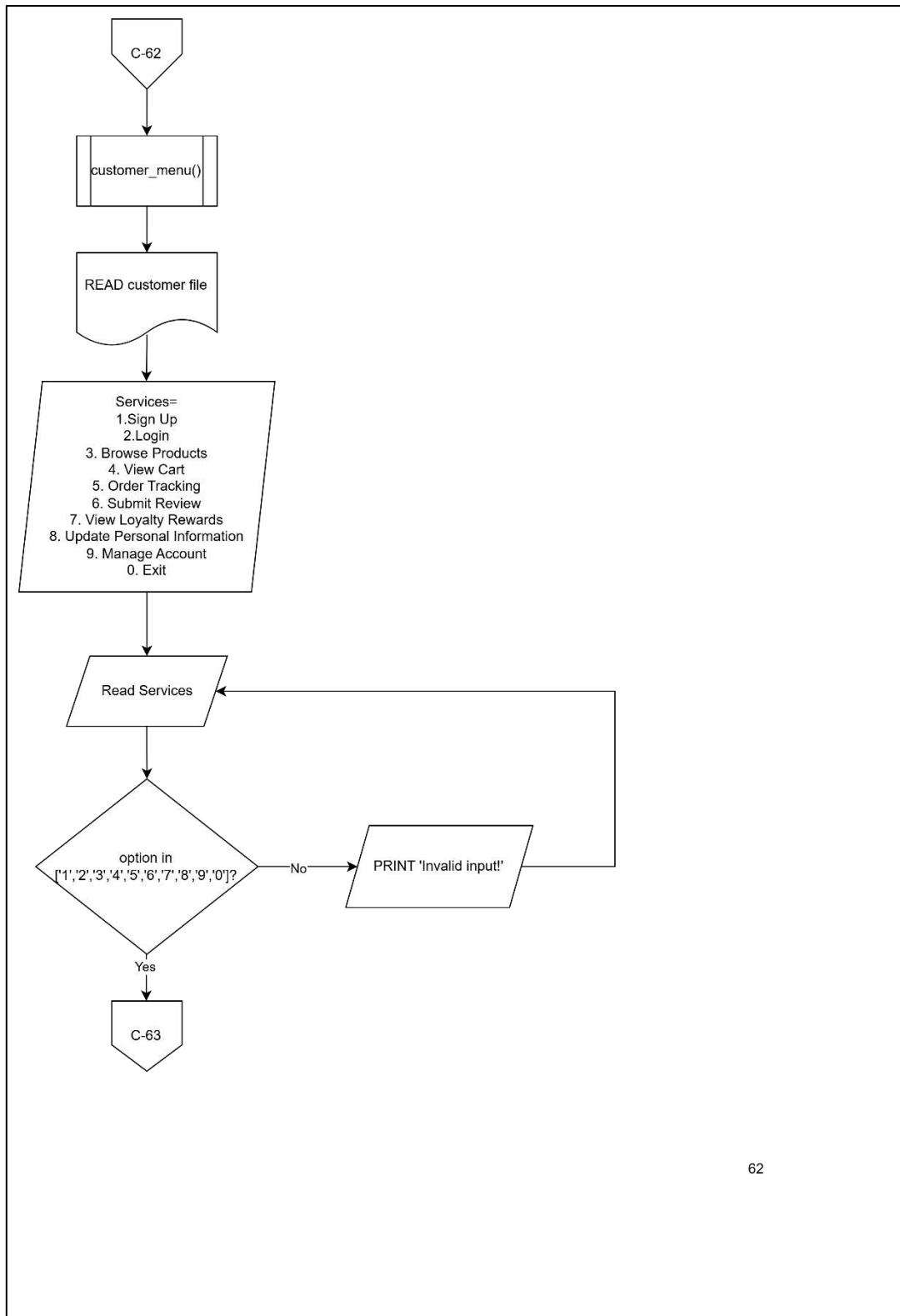
60

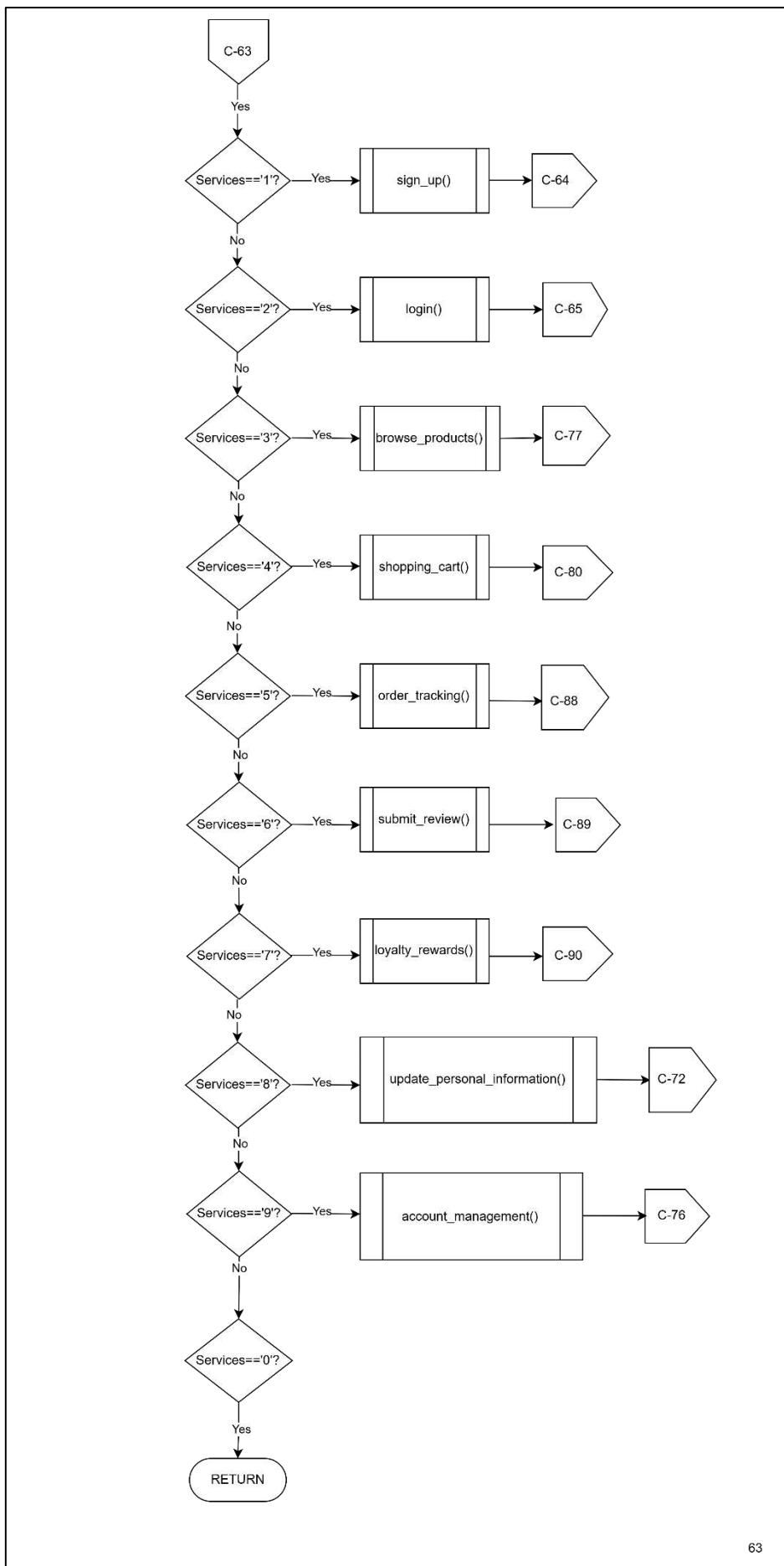


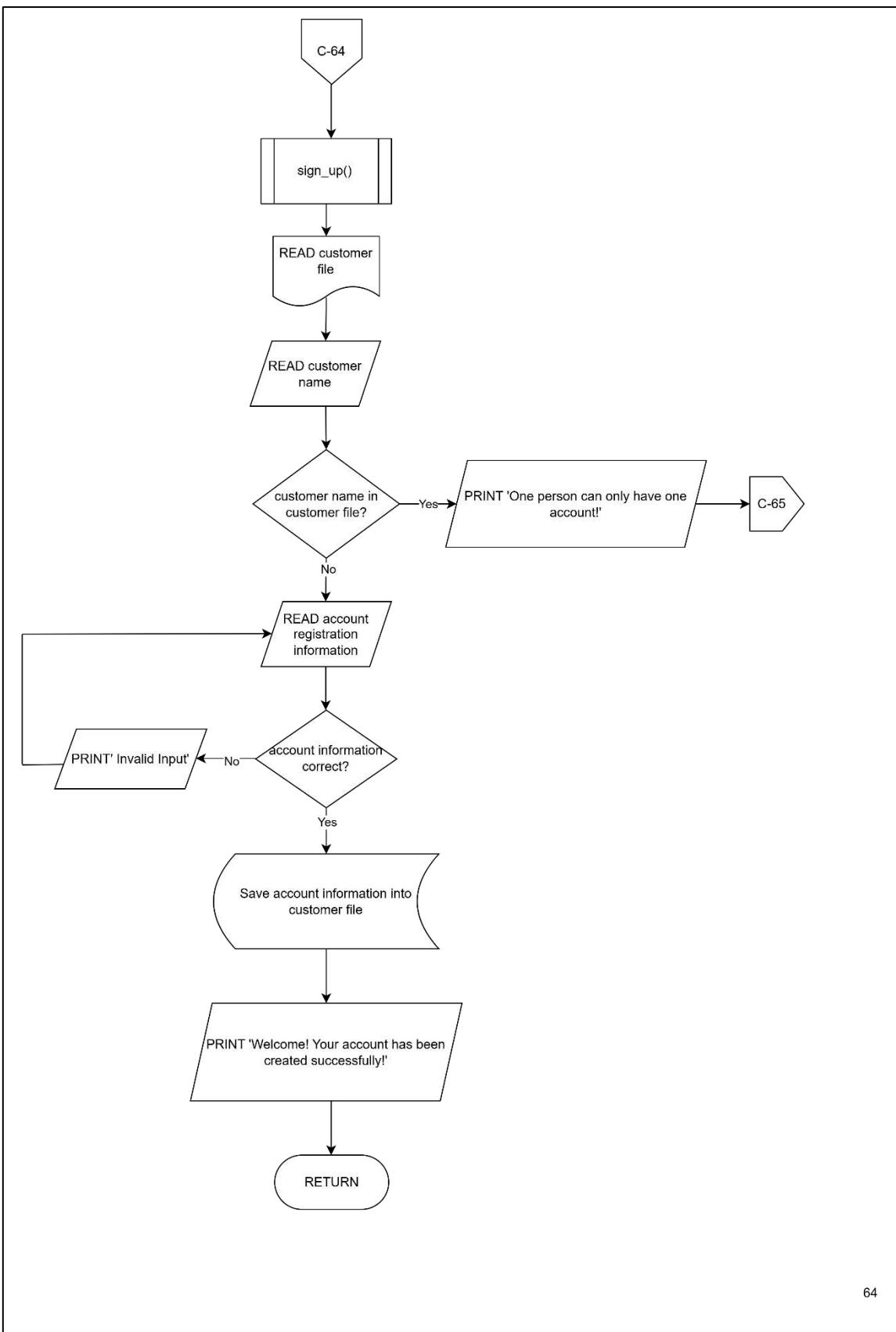
61

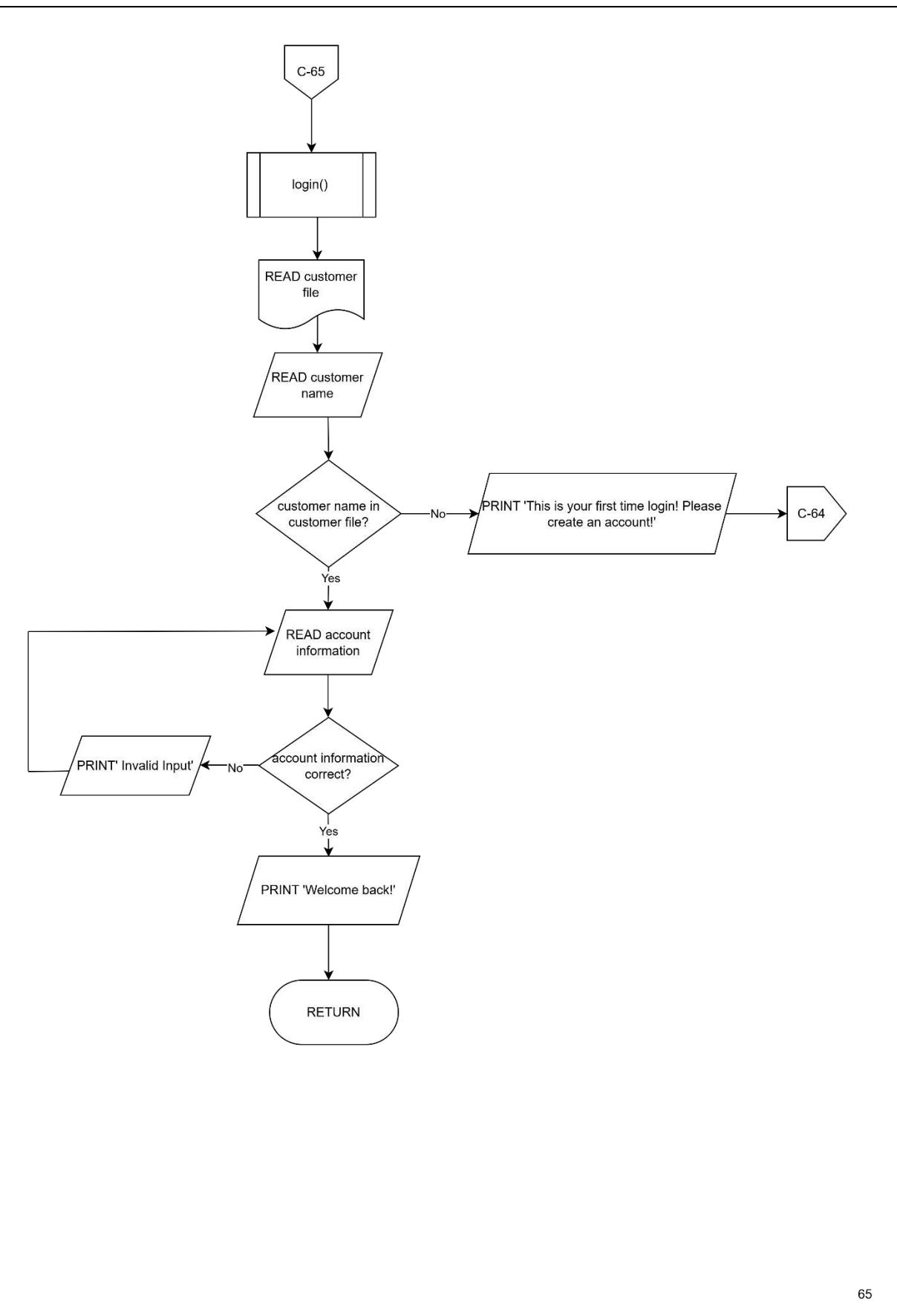
3.3 Customer

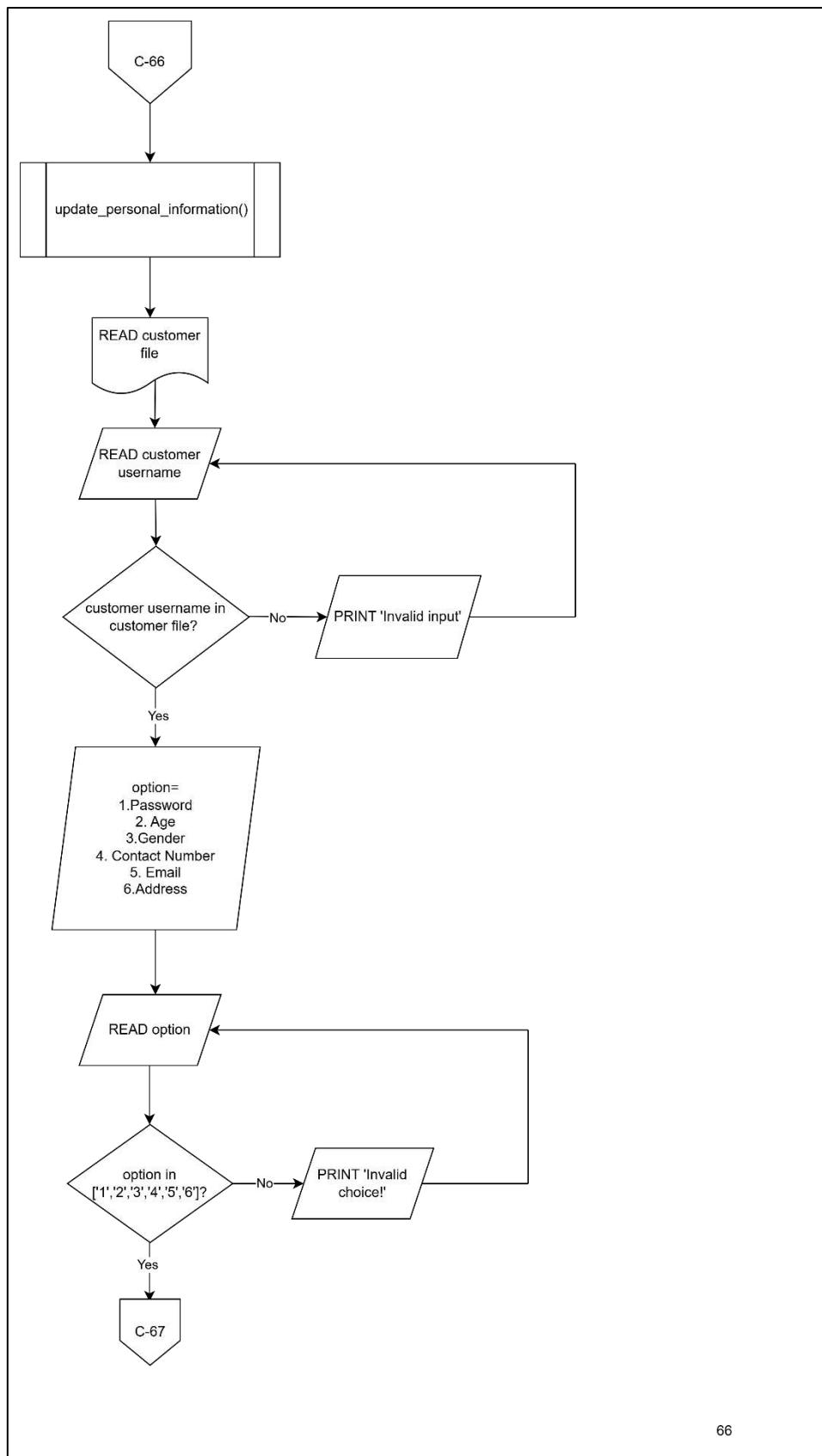
3.3.1 Customer Account Management

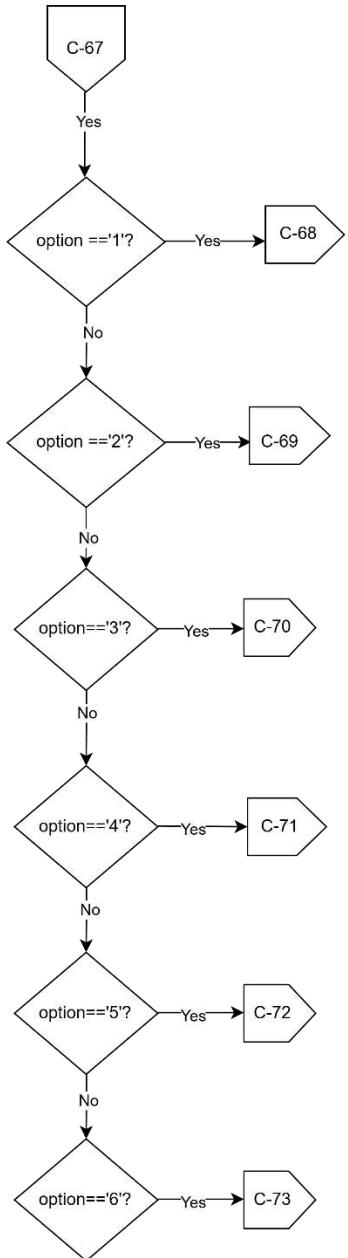


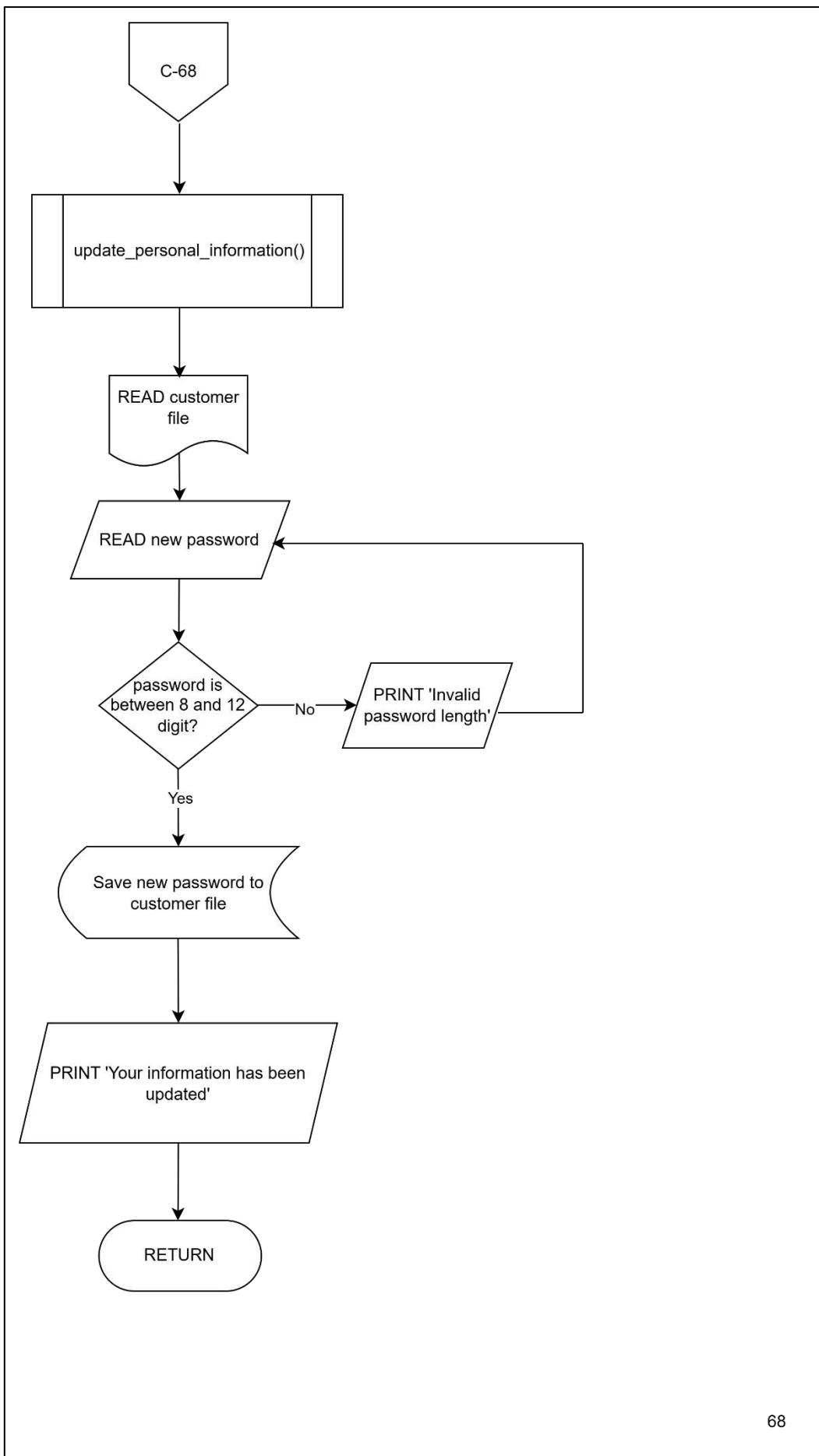


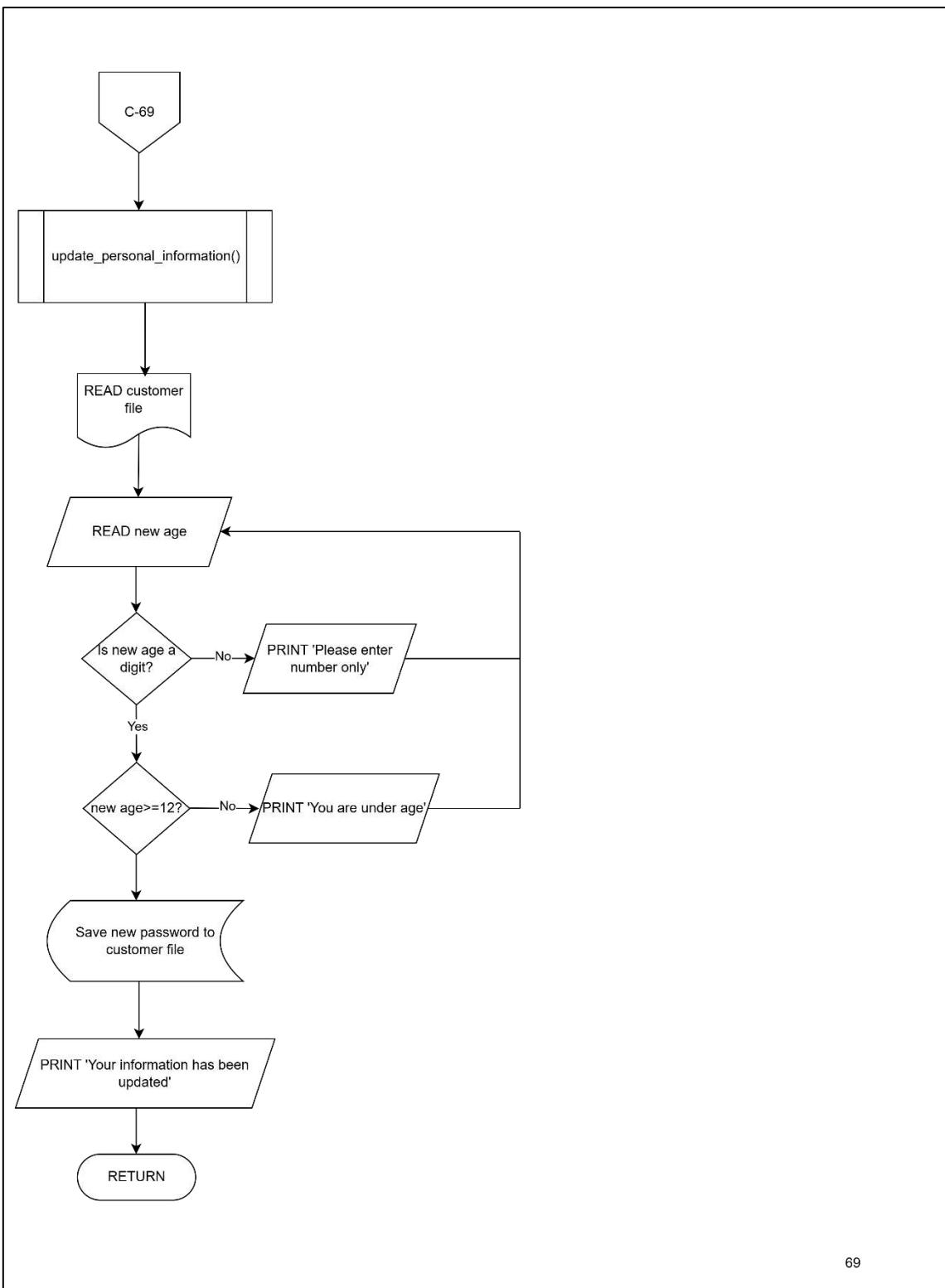




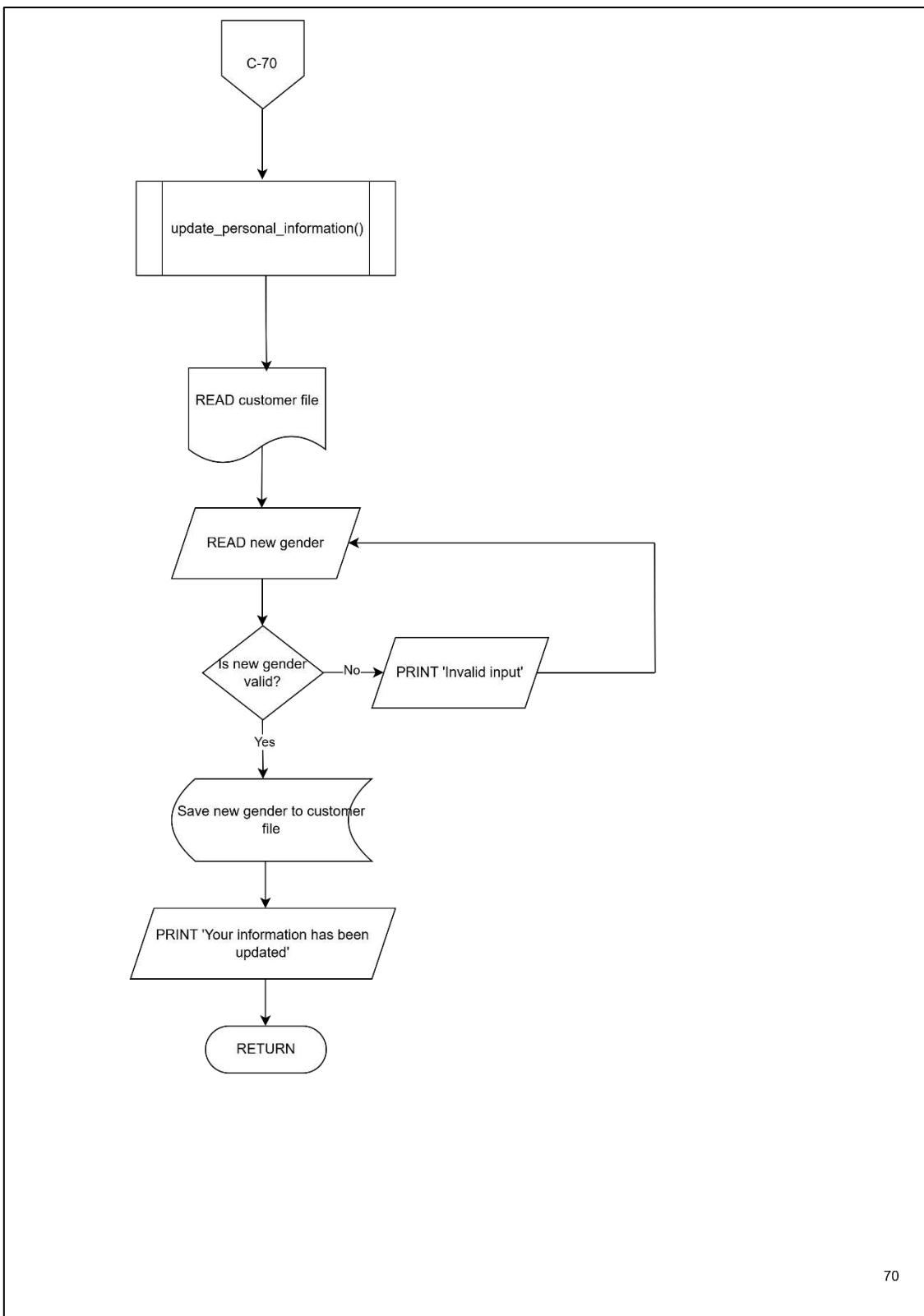


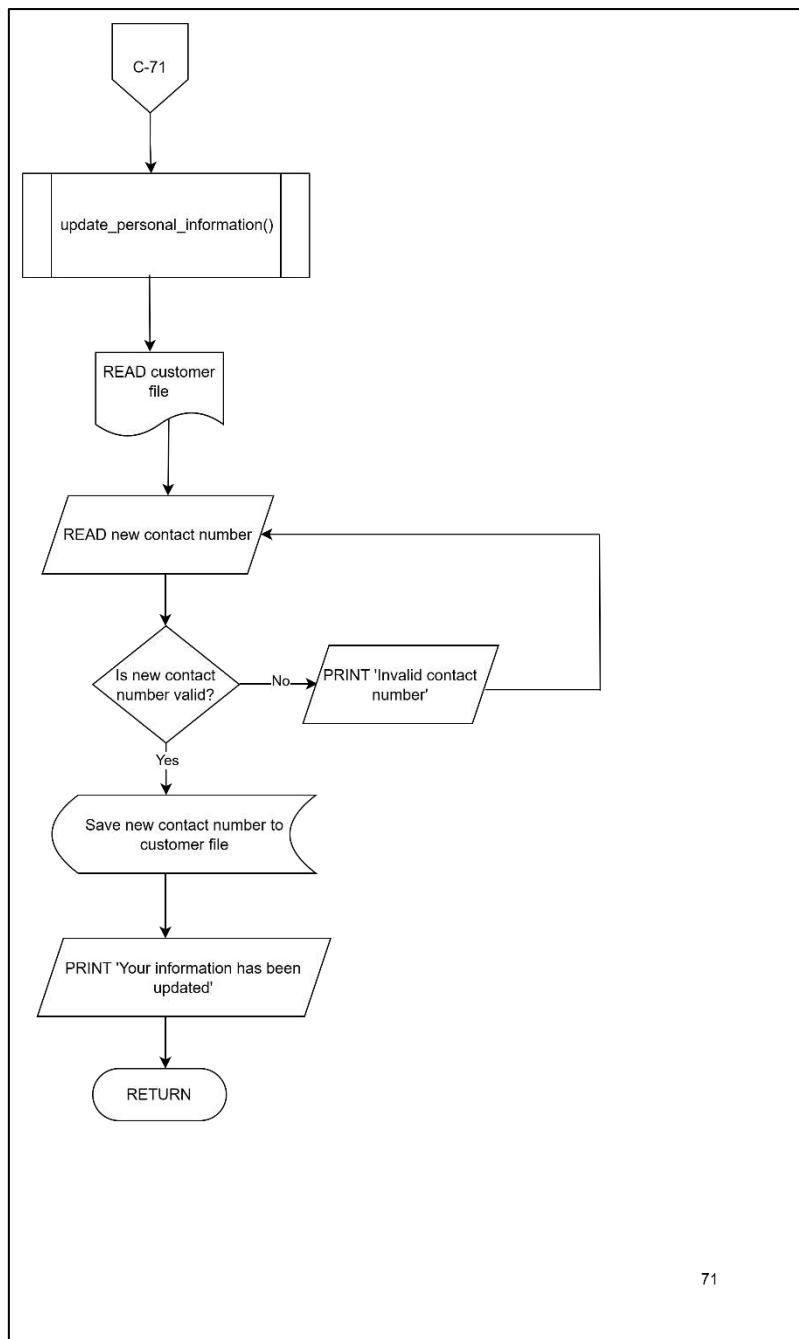




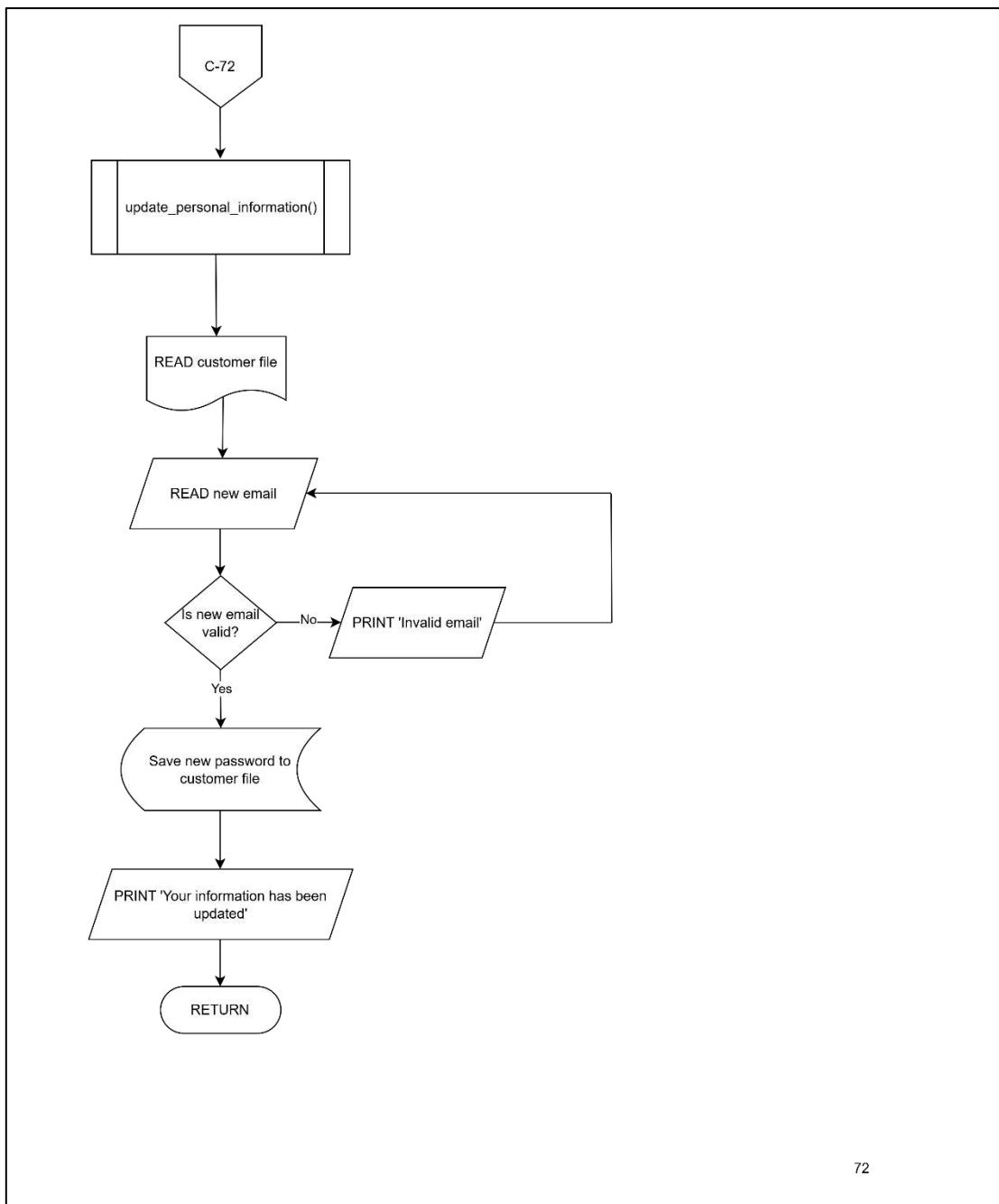


69

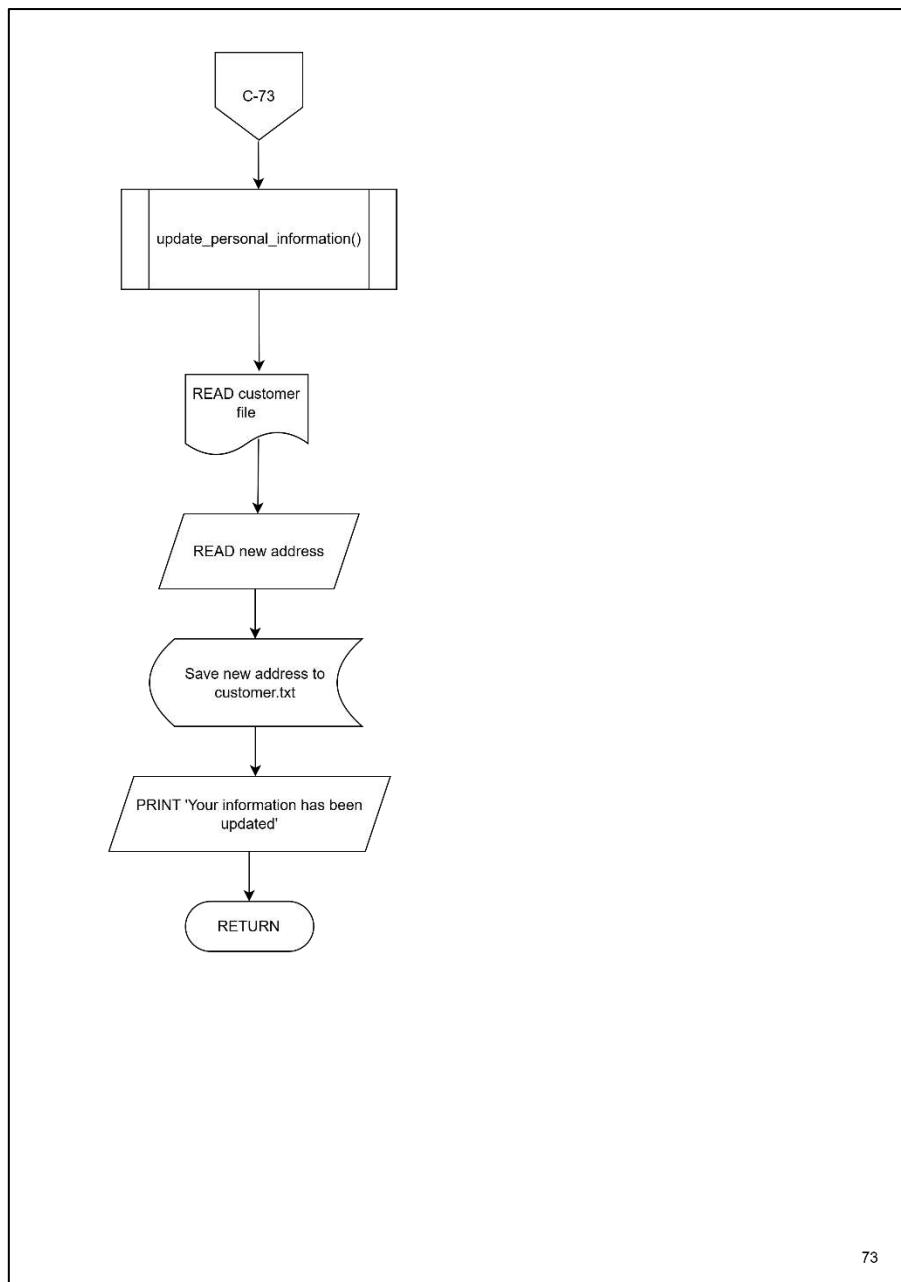




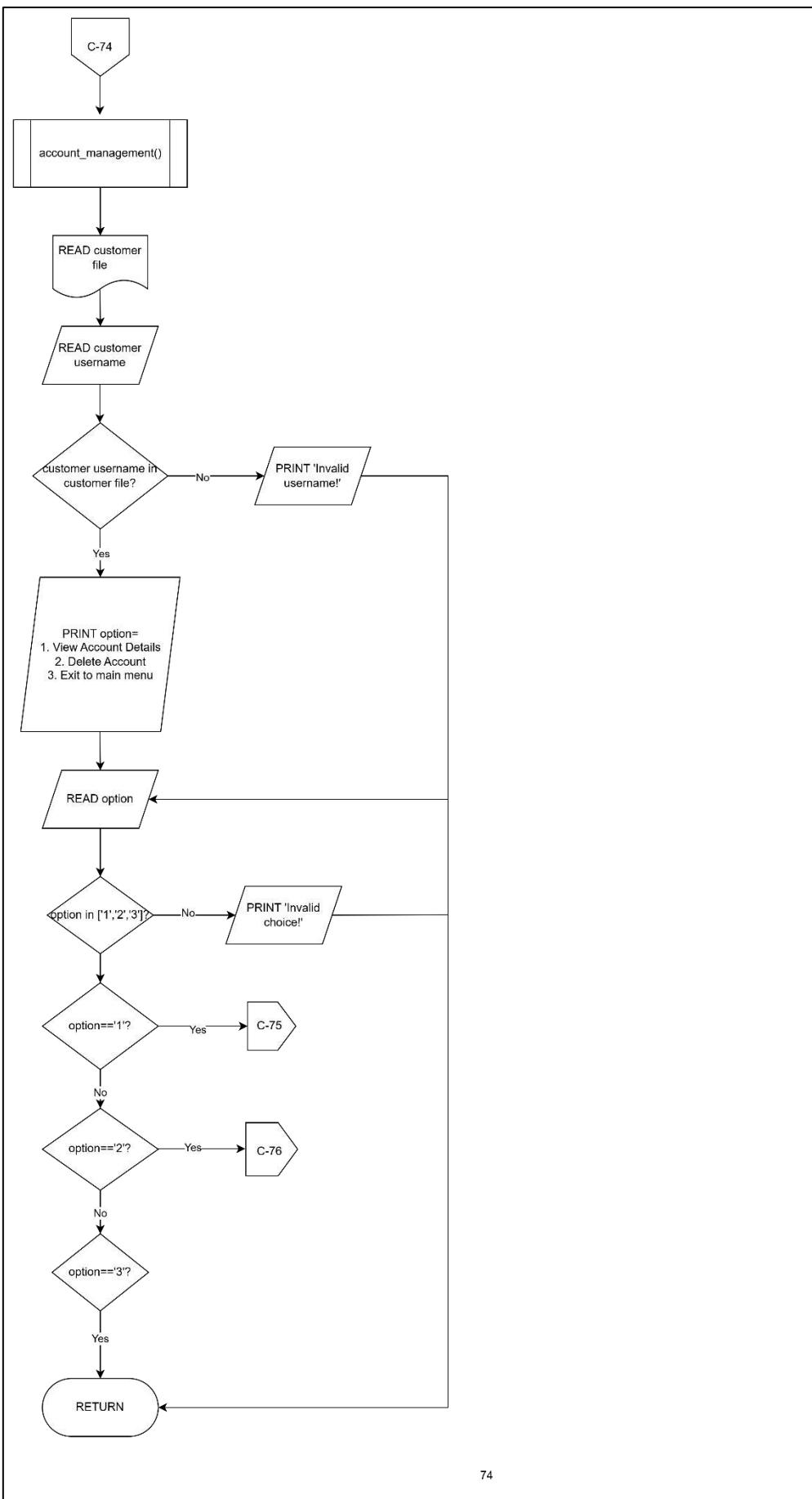
71

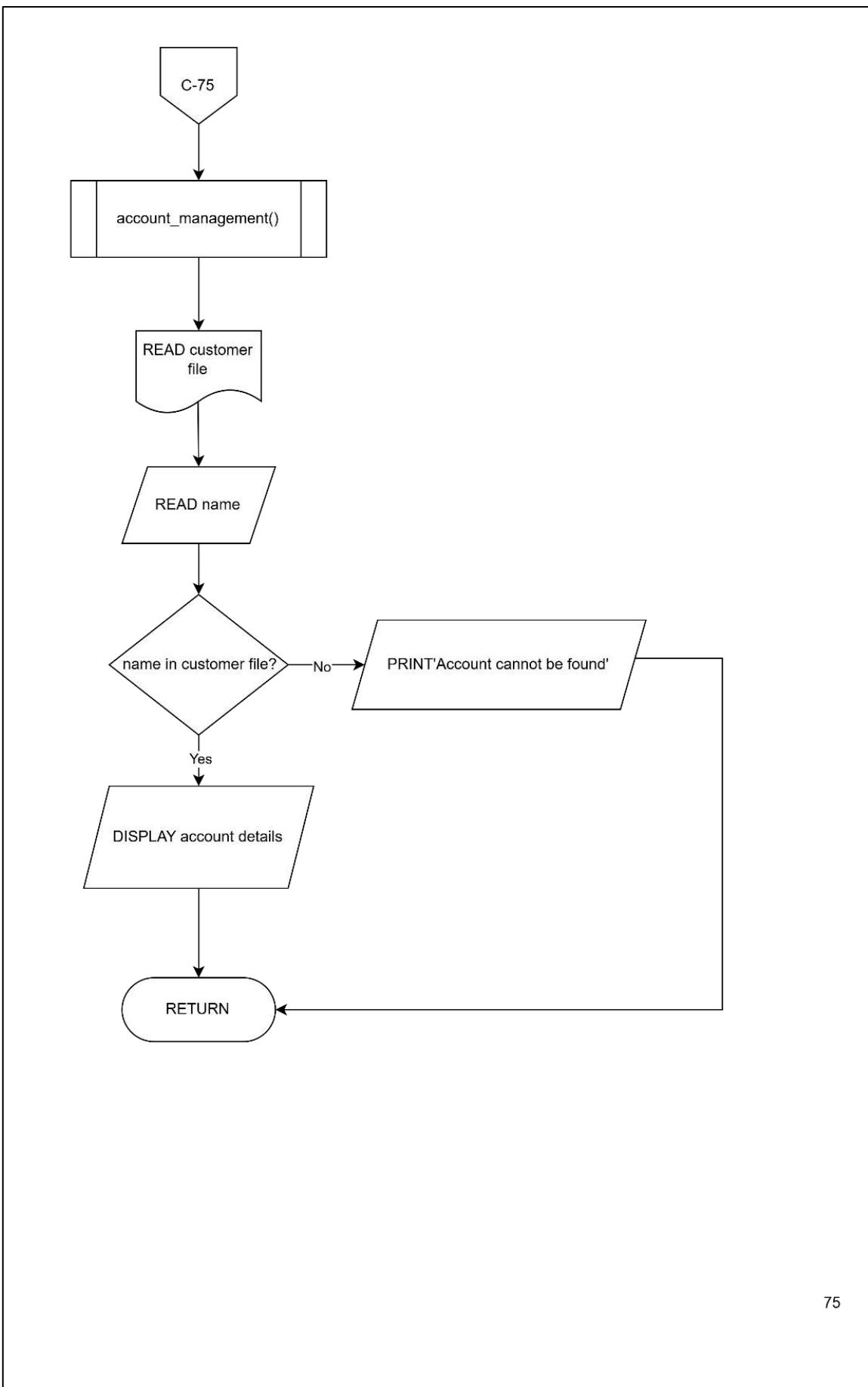


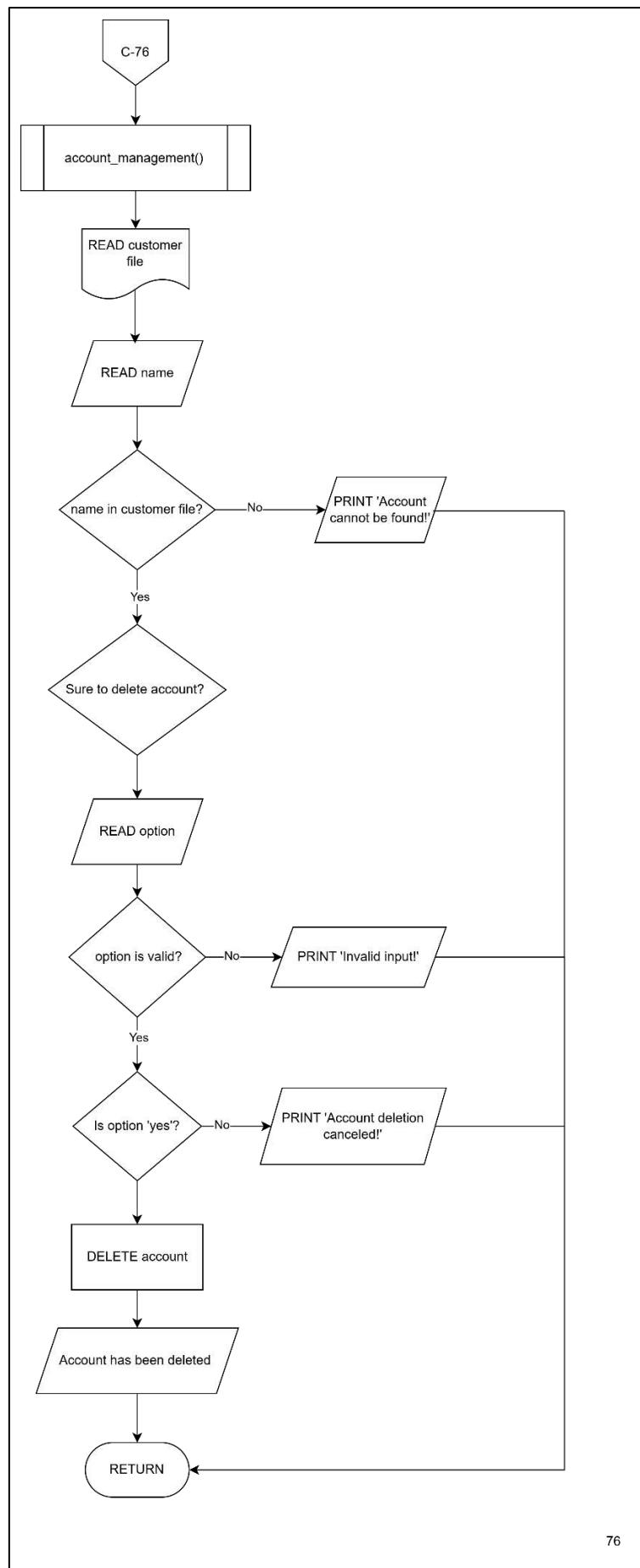
72



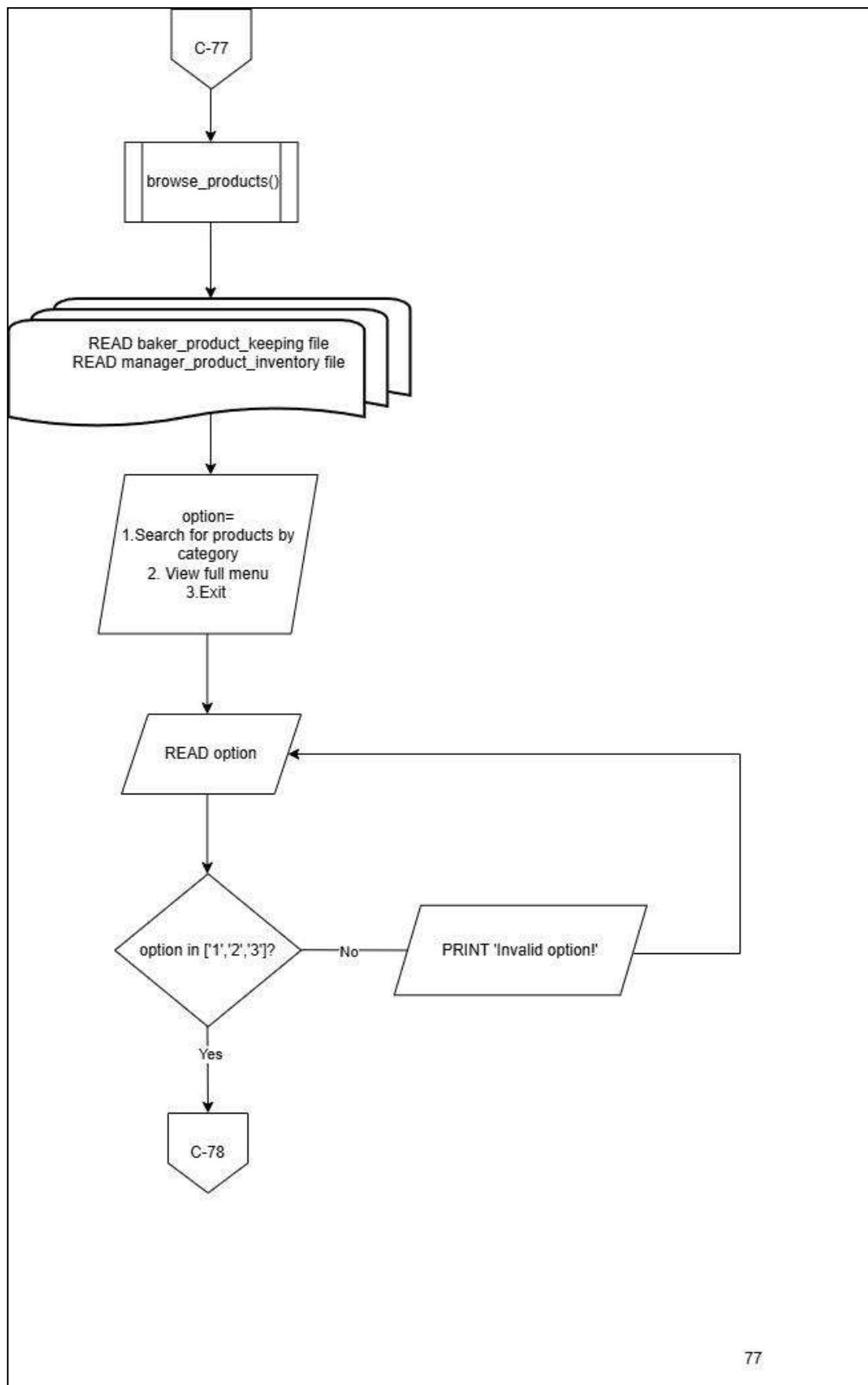
73

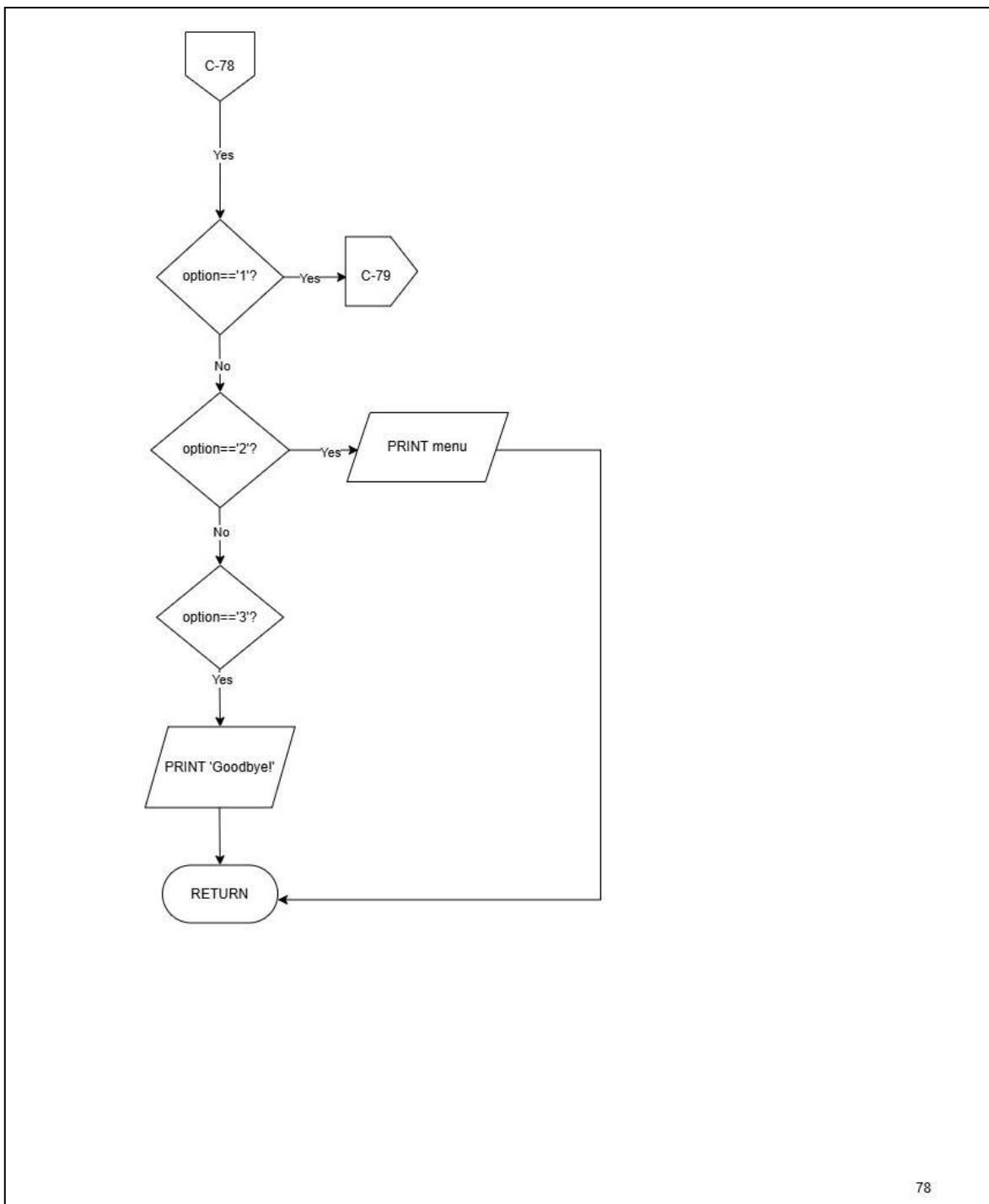




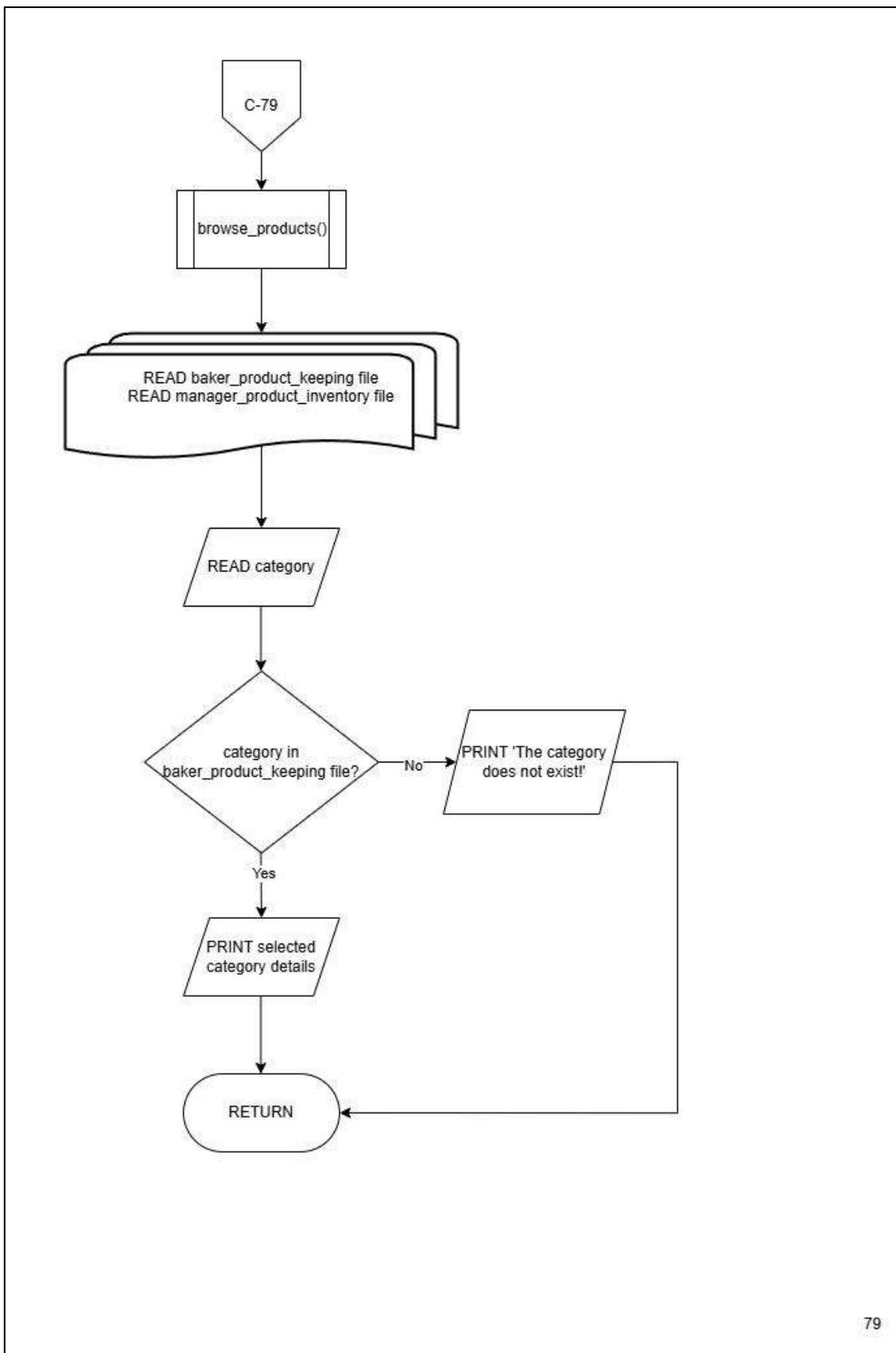


3.3.2 Product Browsing

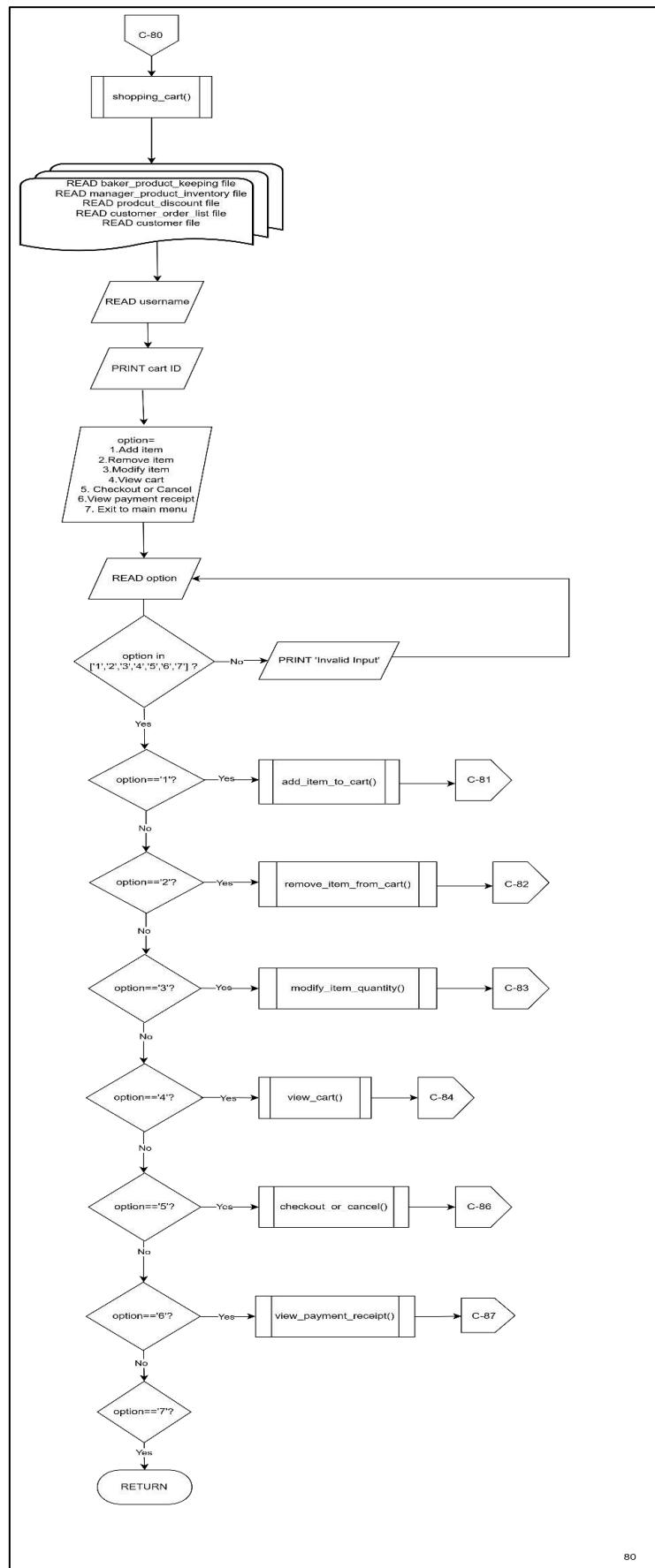


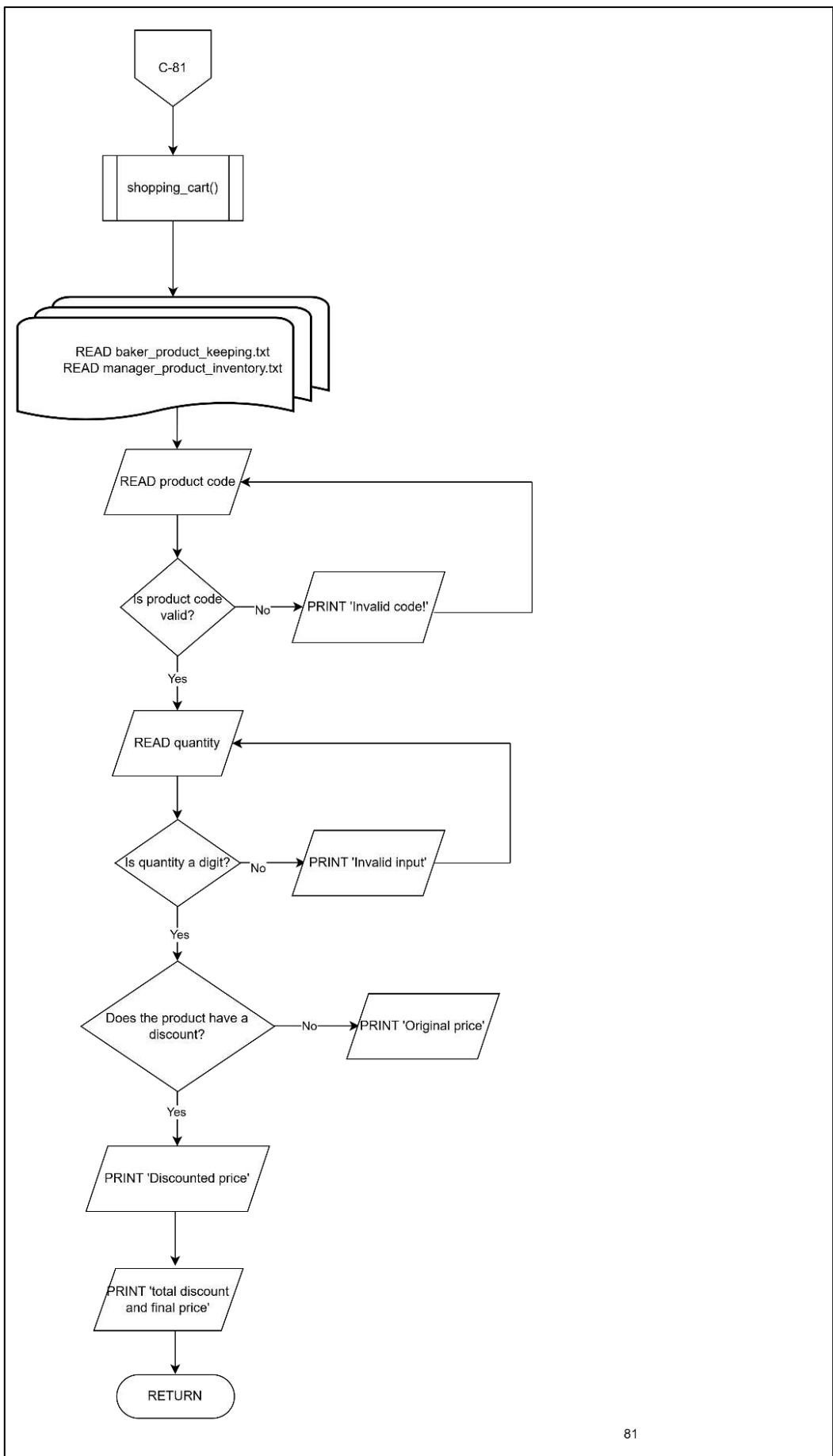


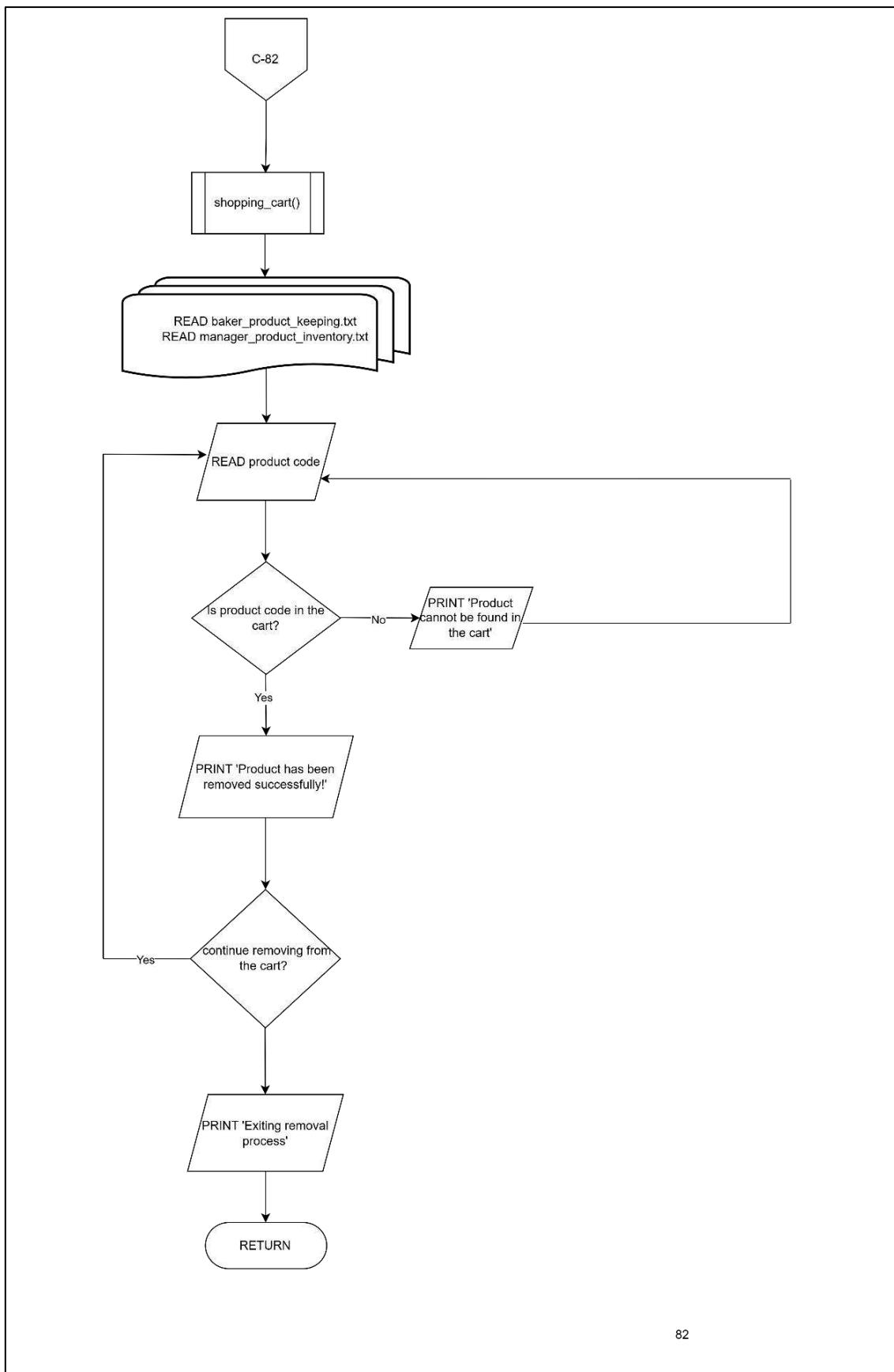
78

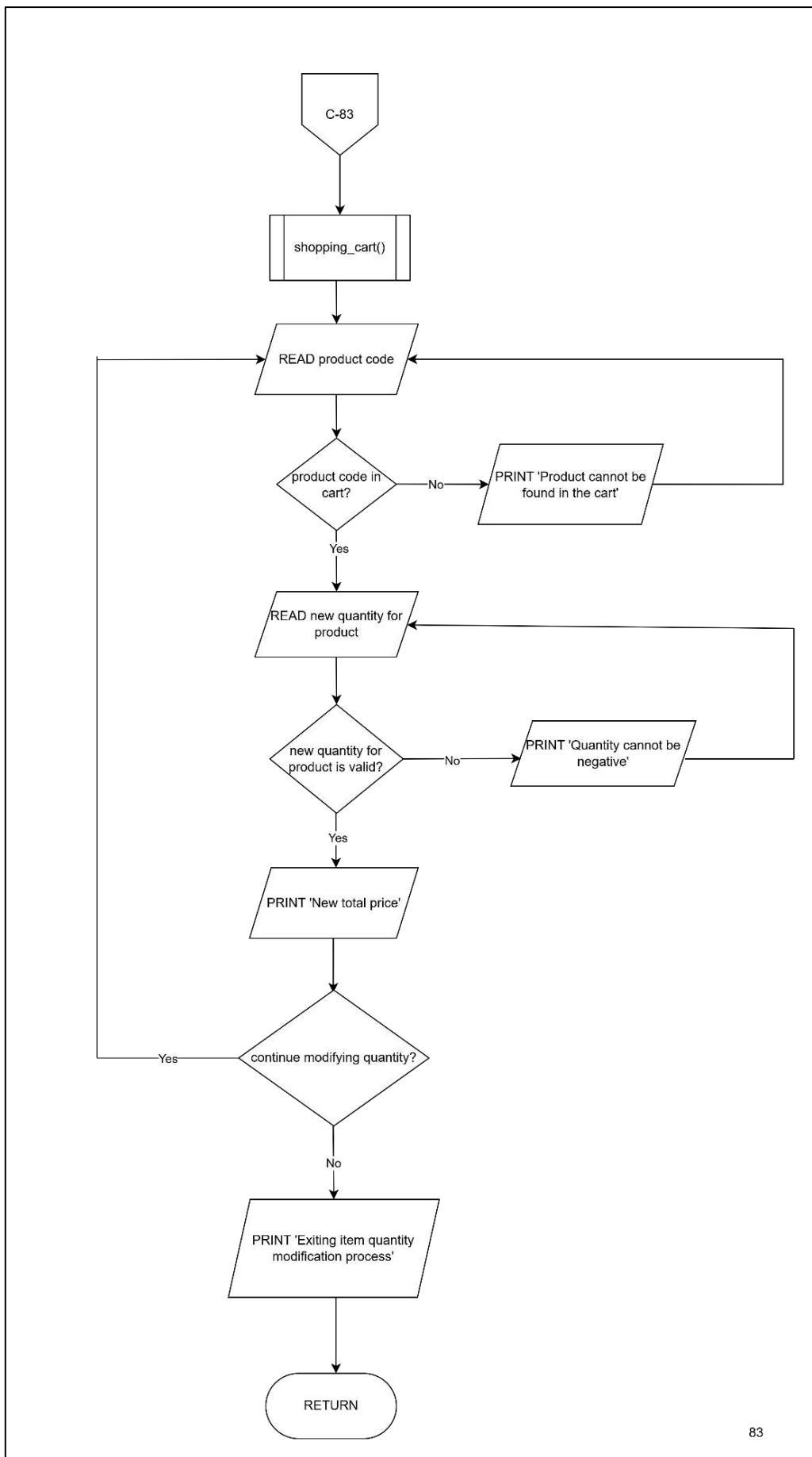


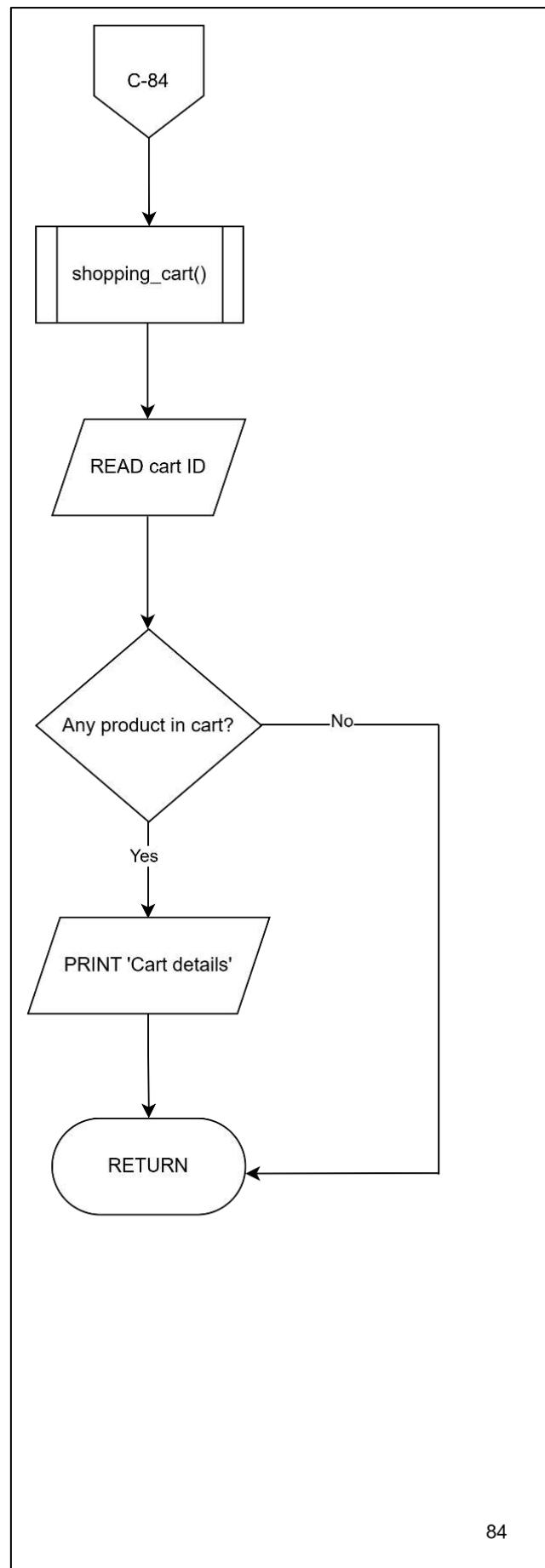
3.3.3 Cart Management

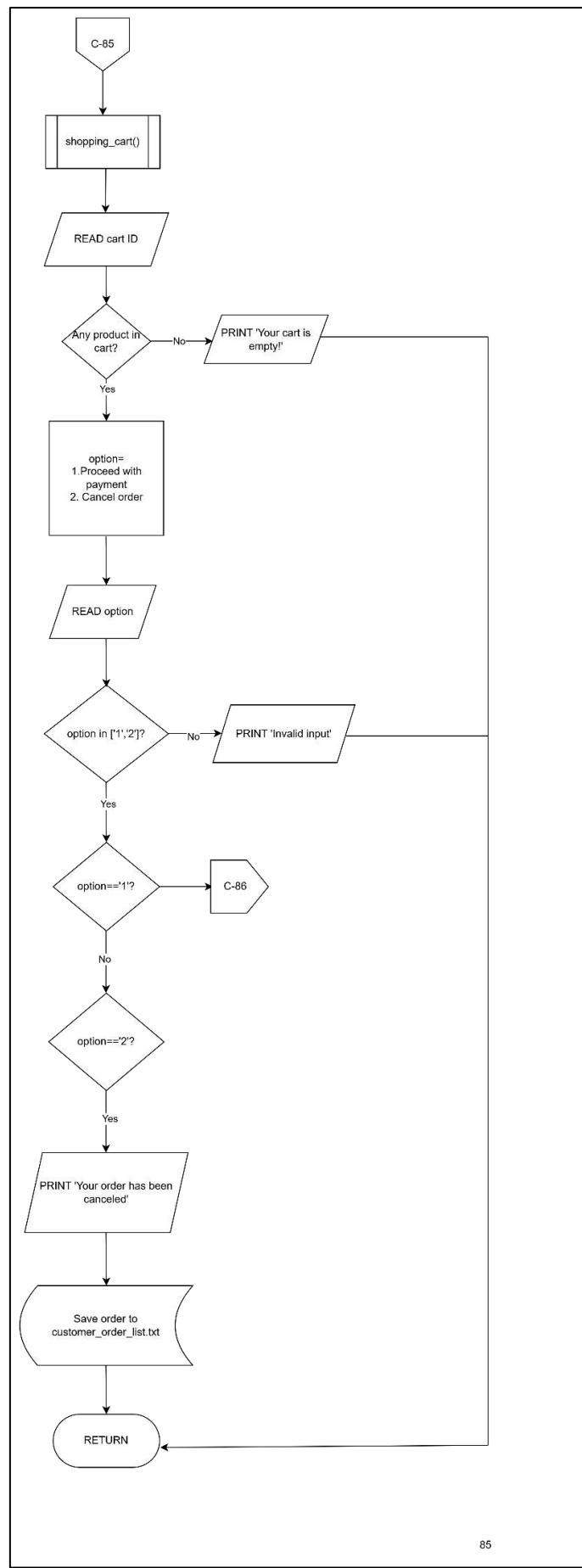


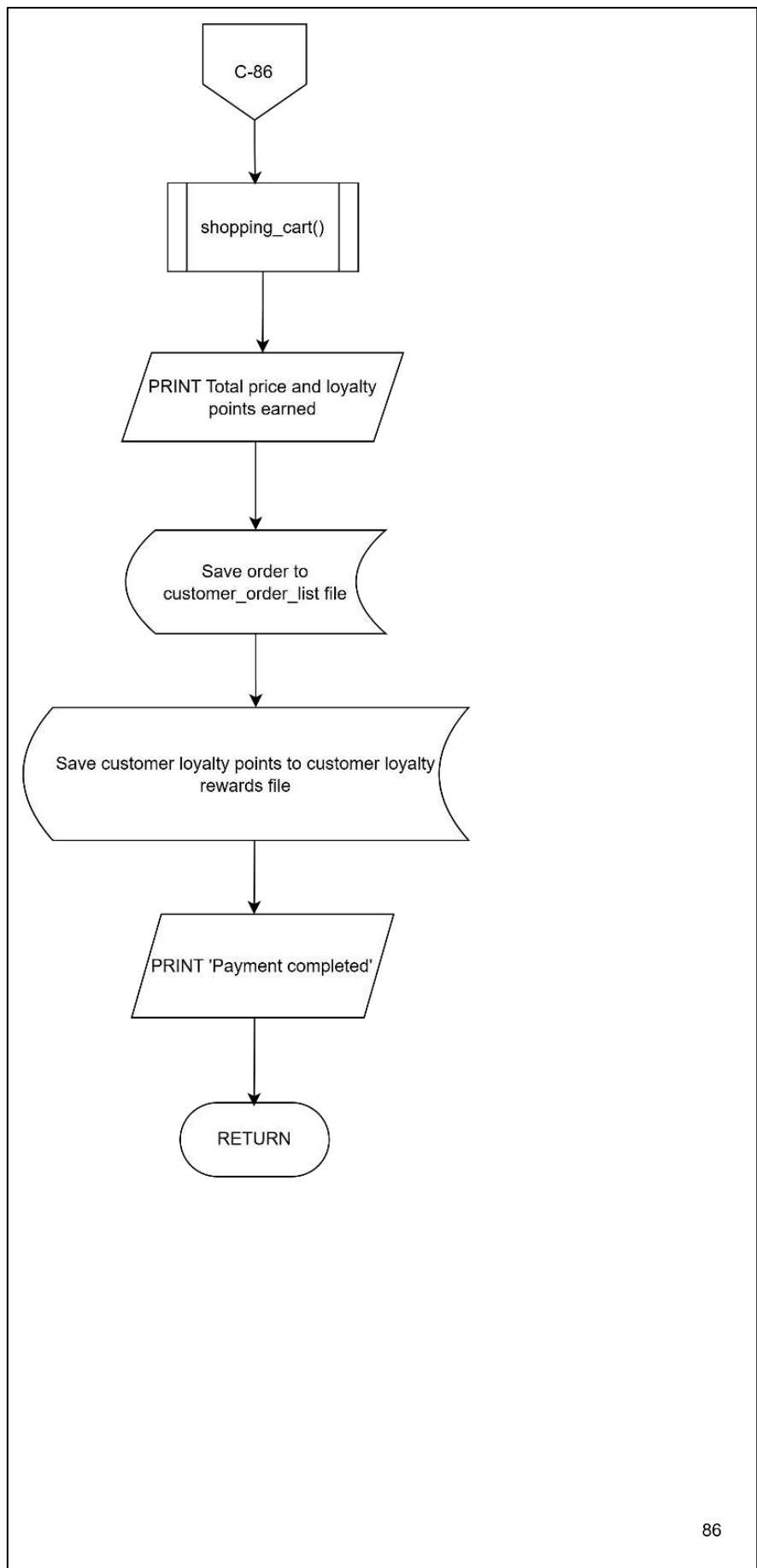


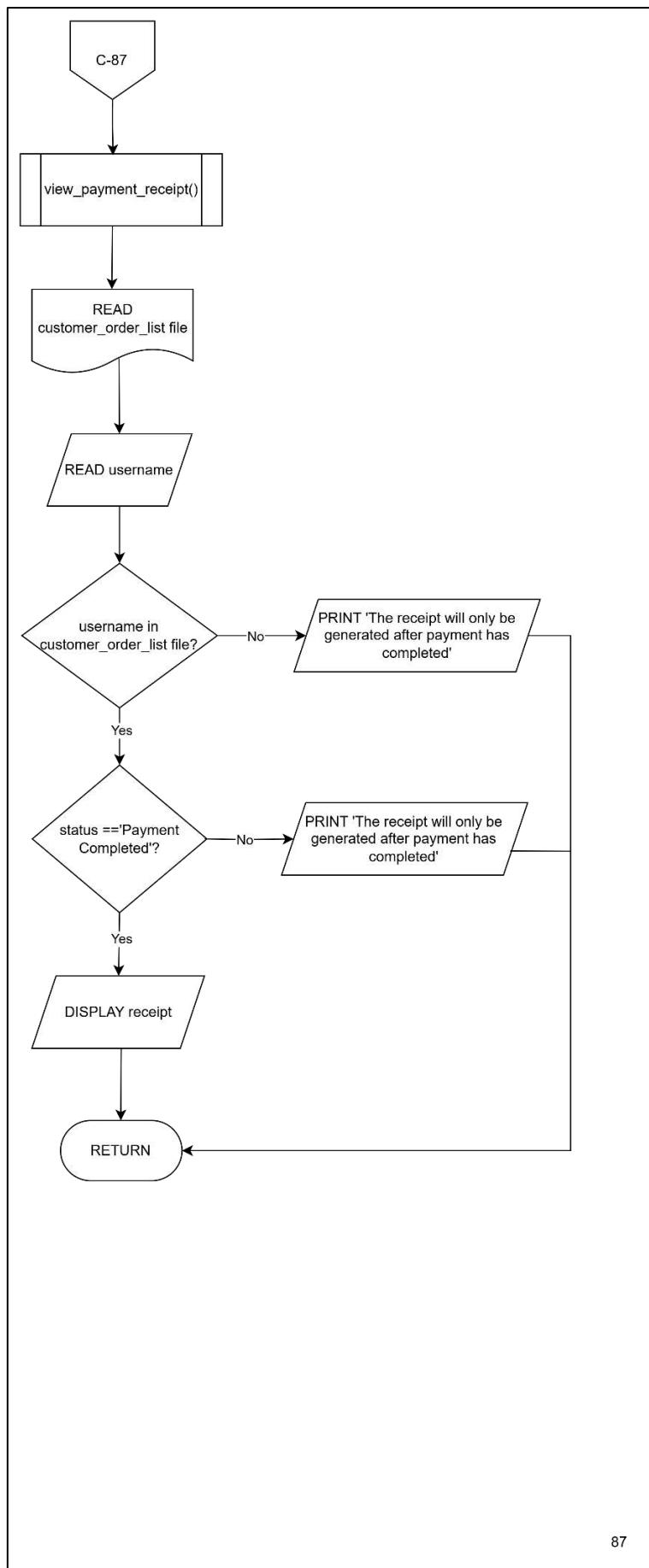




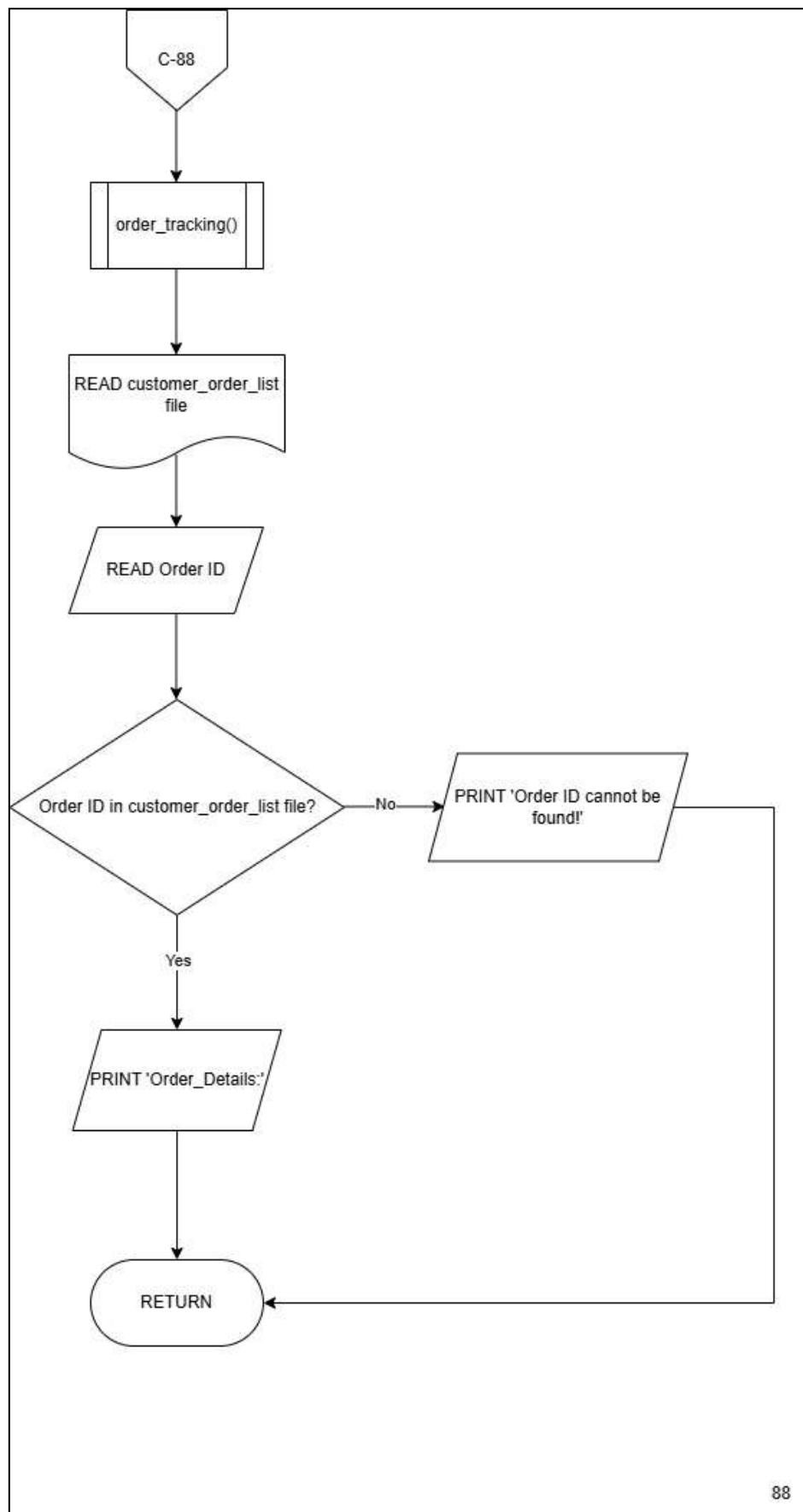




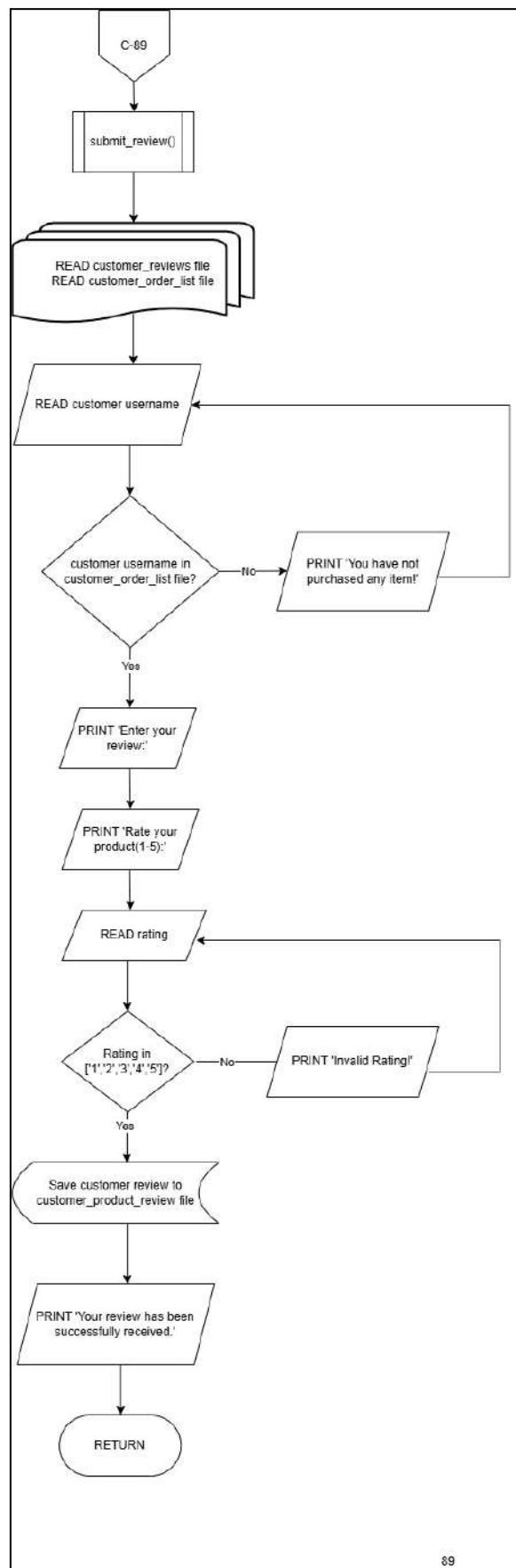




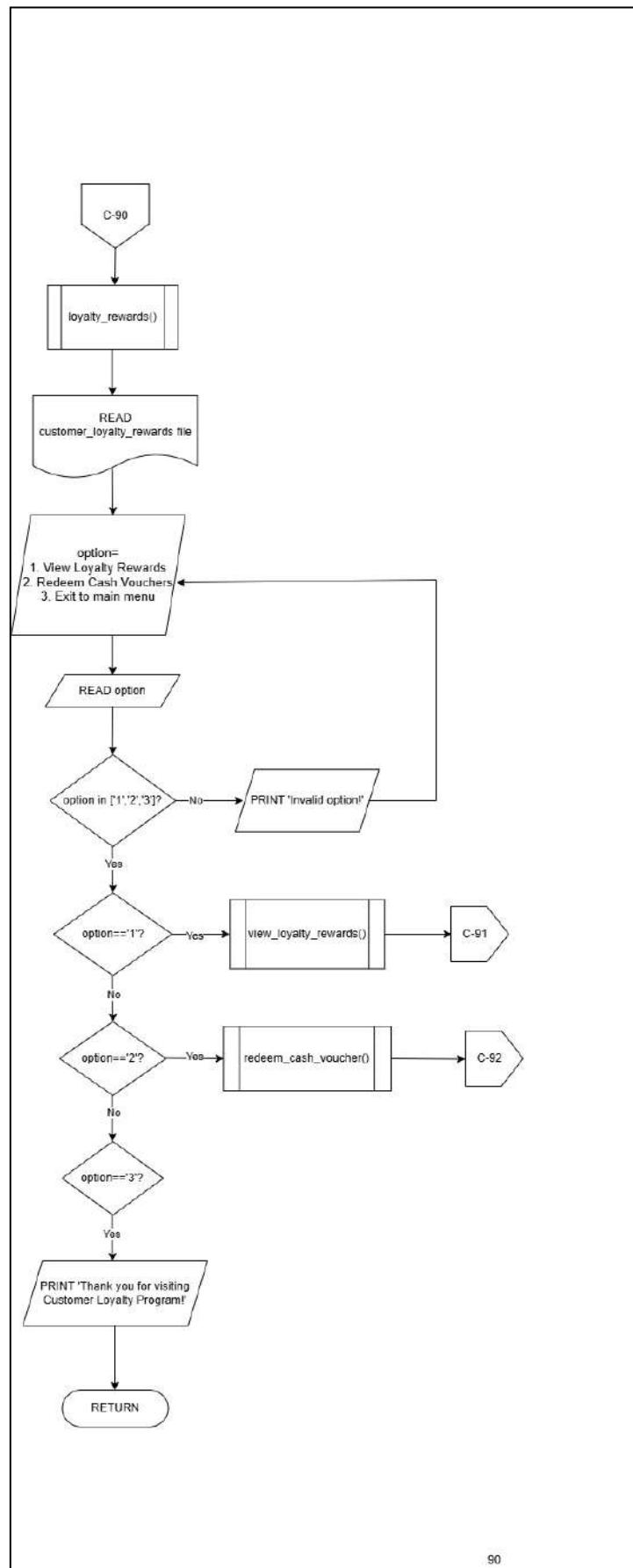
3.3.4 Order Tracking

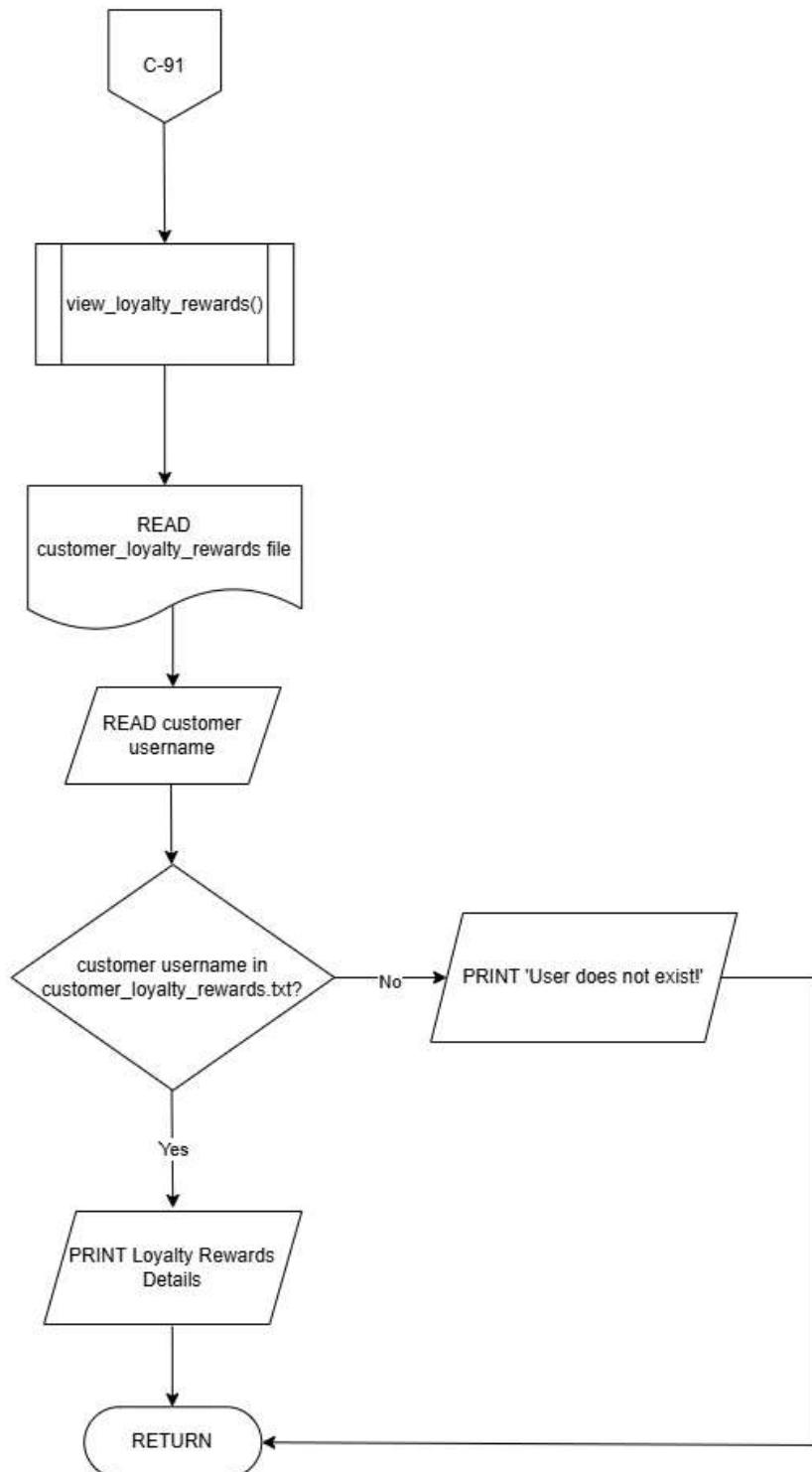


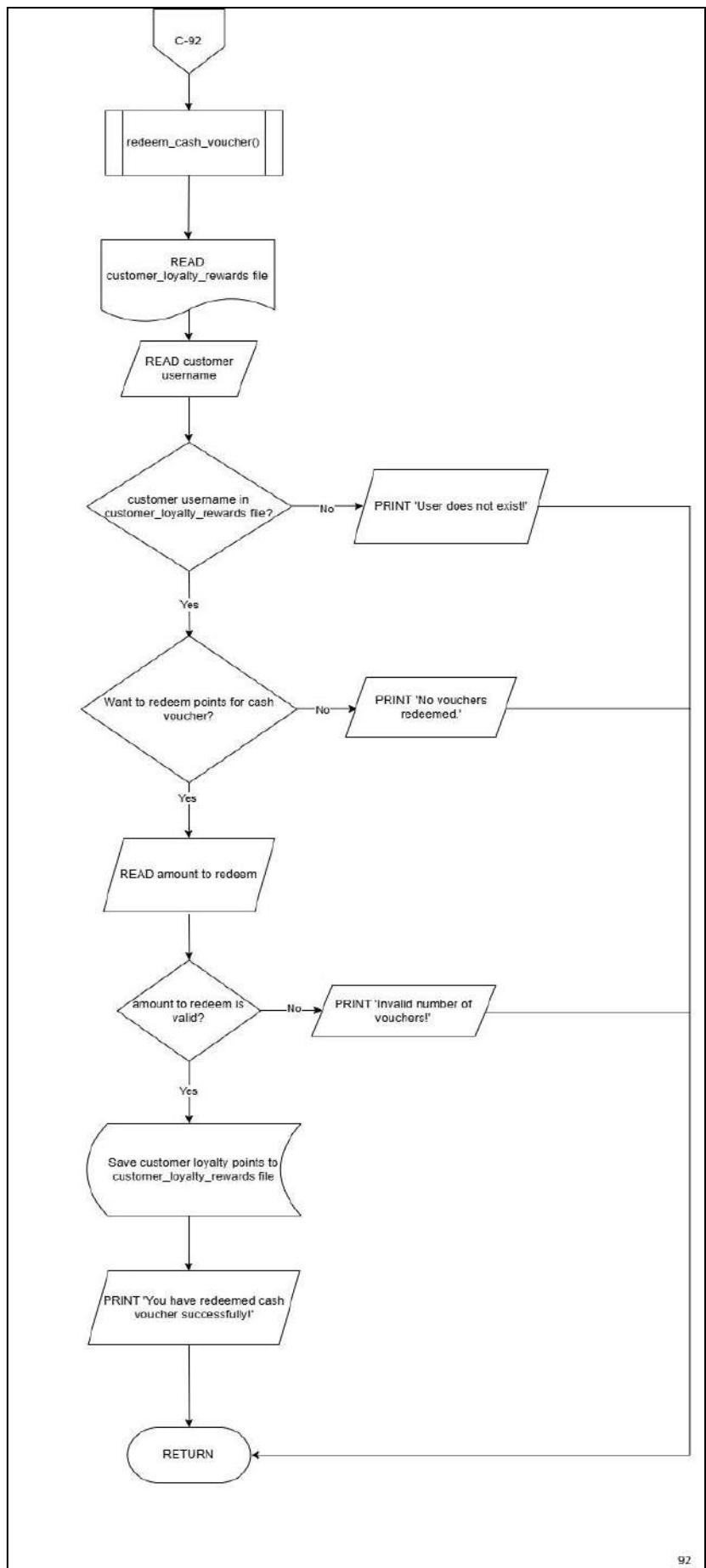
3.3.5 Product Review



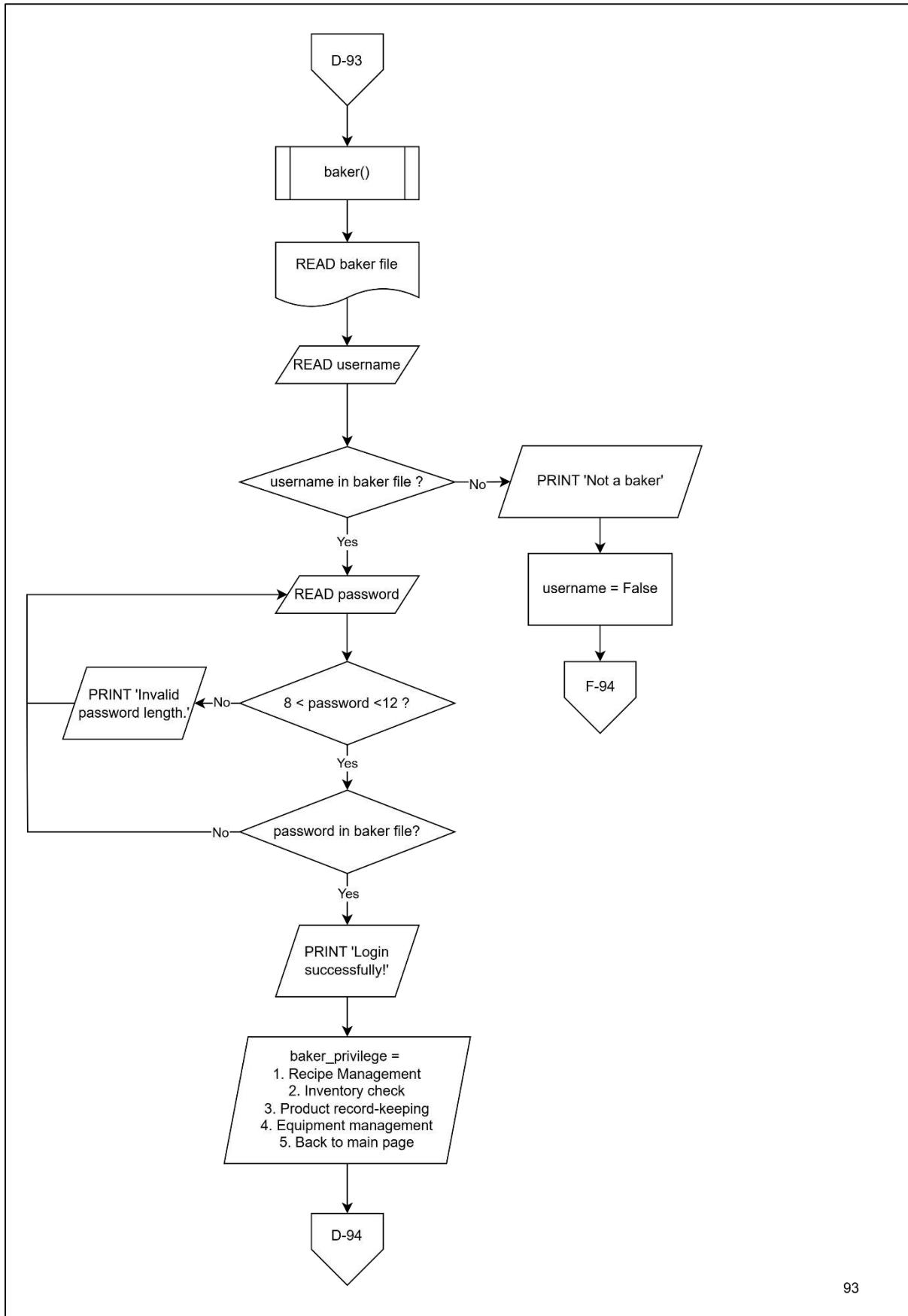
3.3.6 Customer Loyalty Rewards

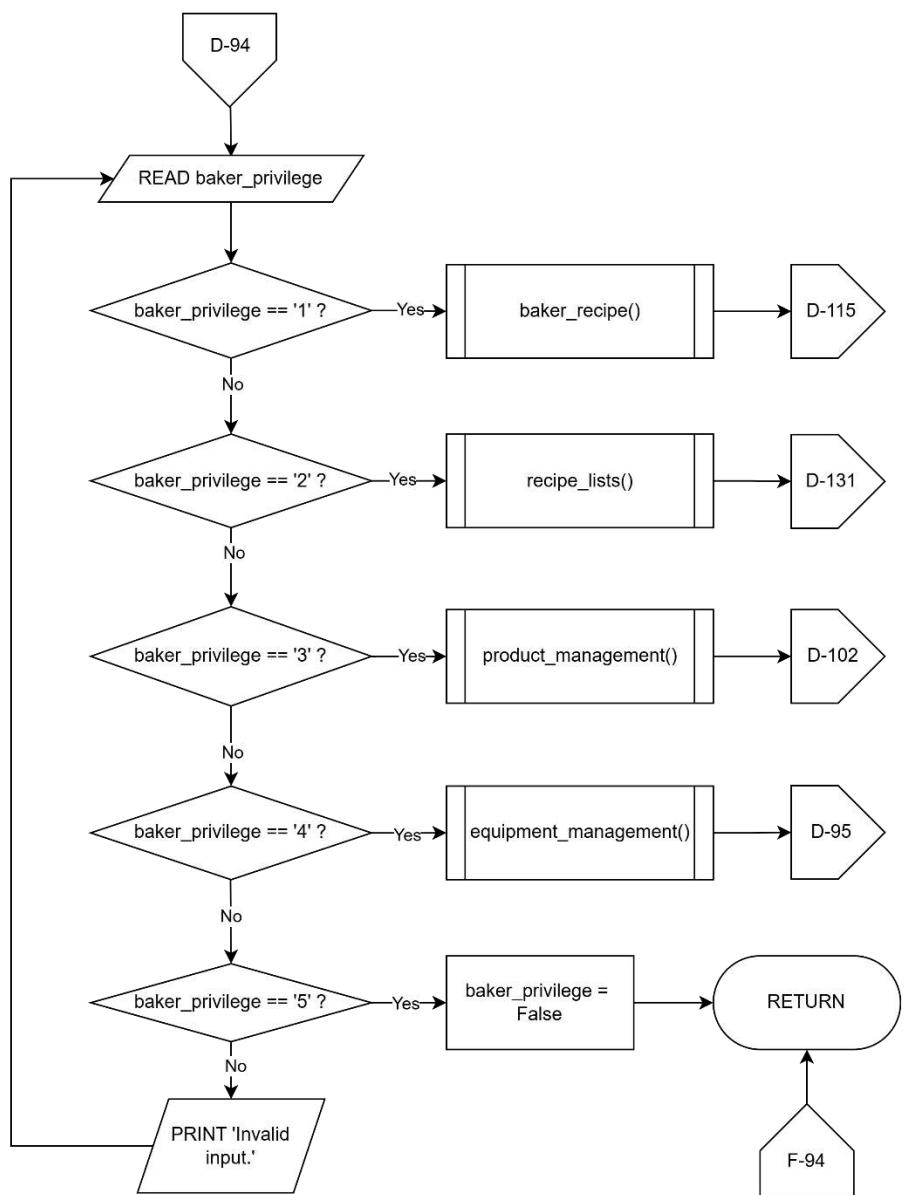




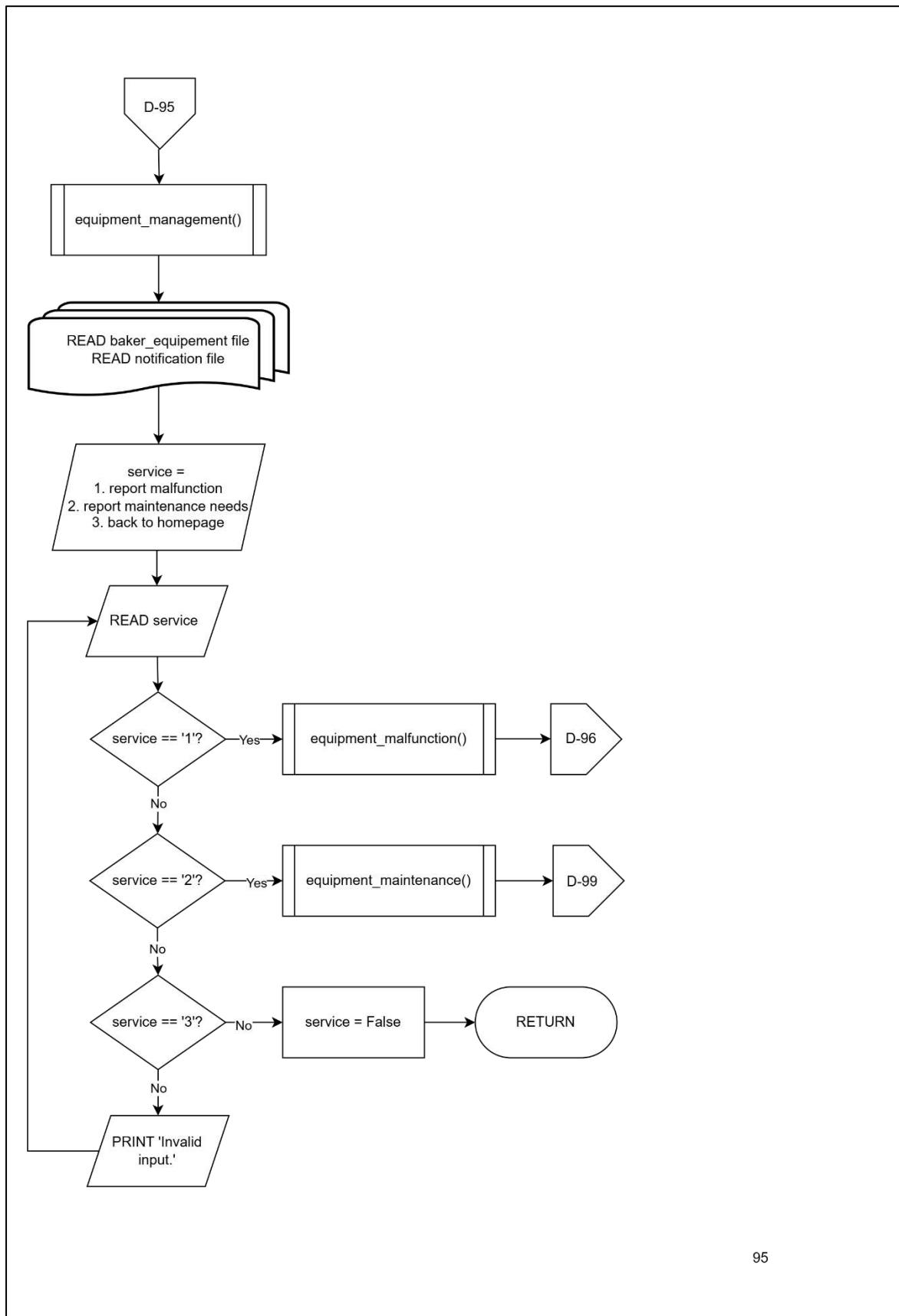


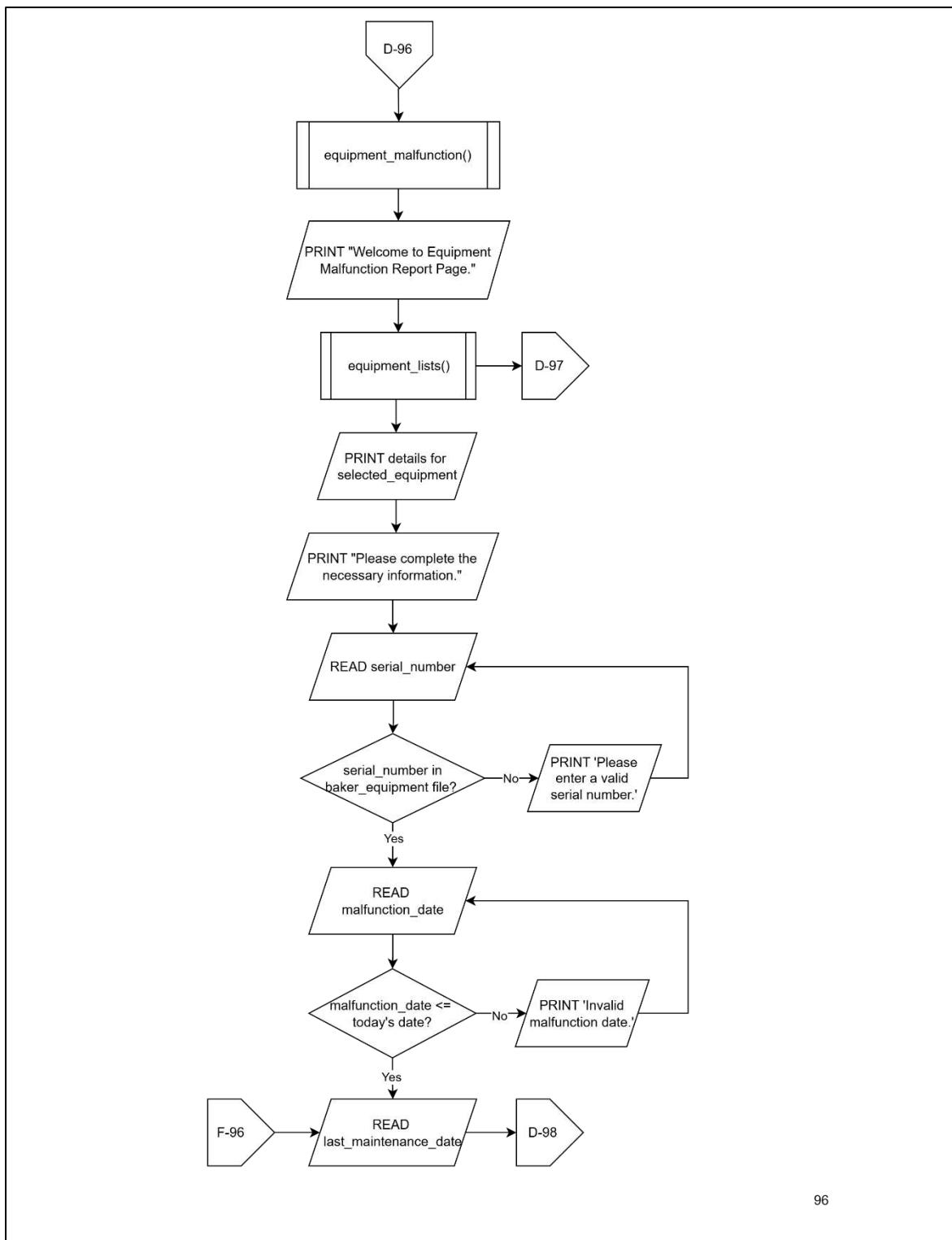
3.5 Baker(s)

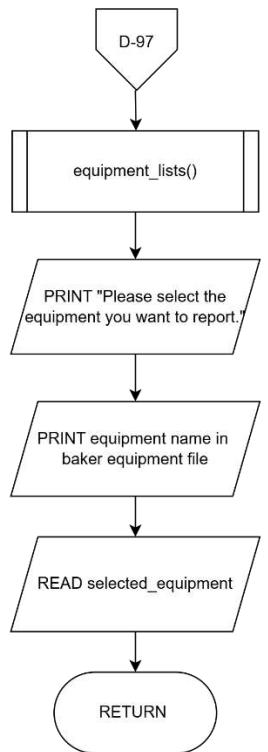


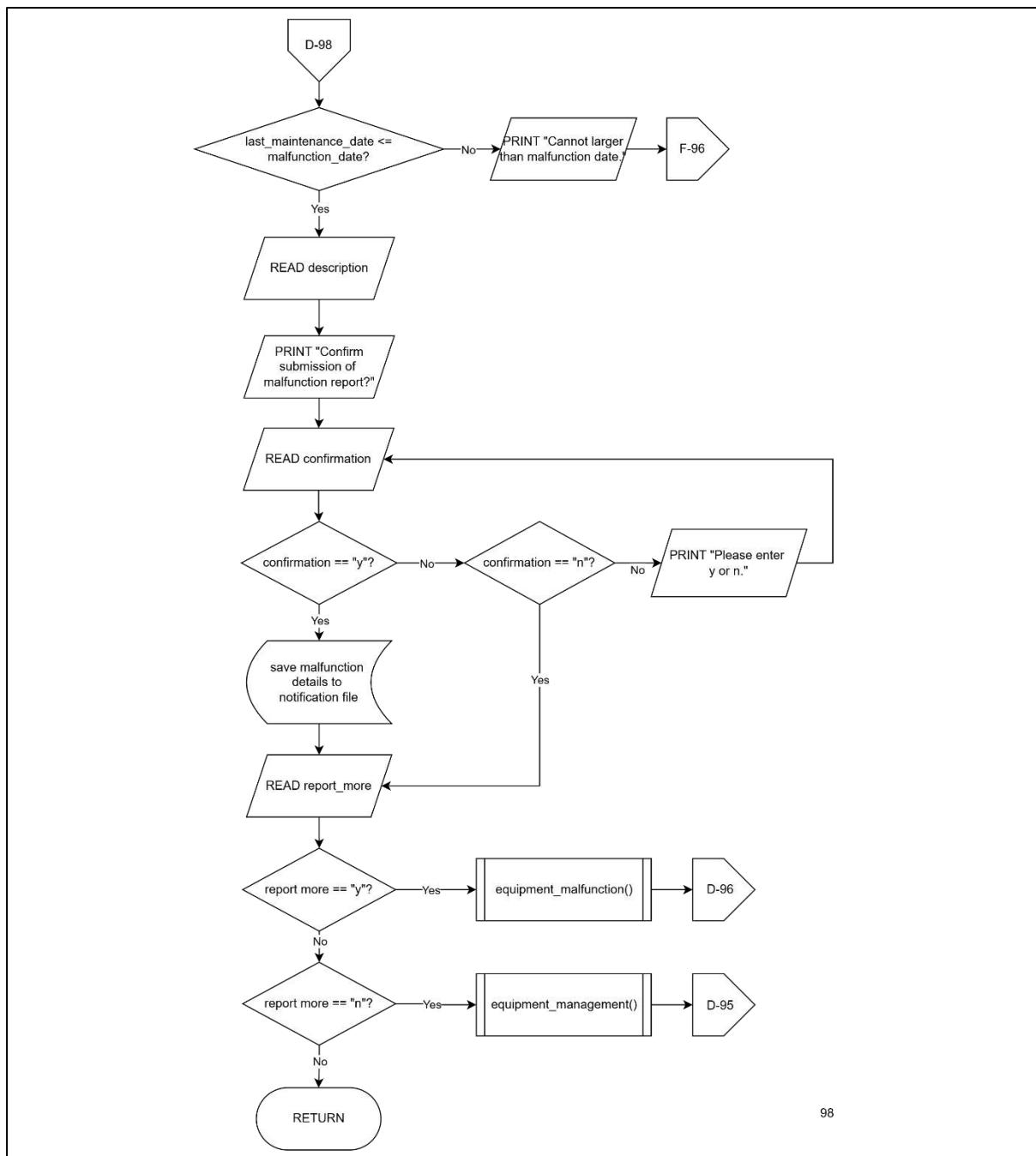


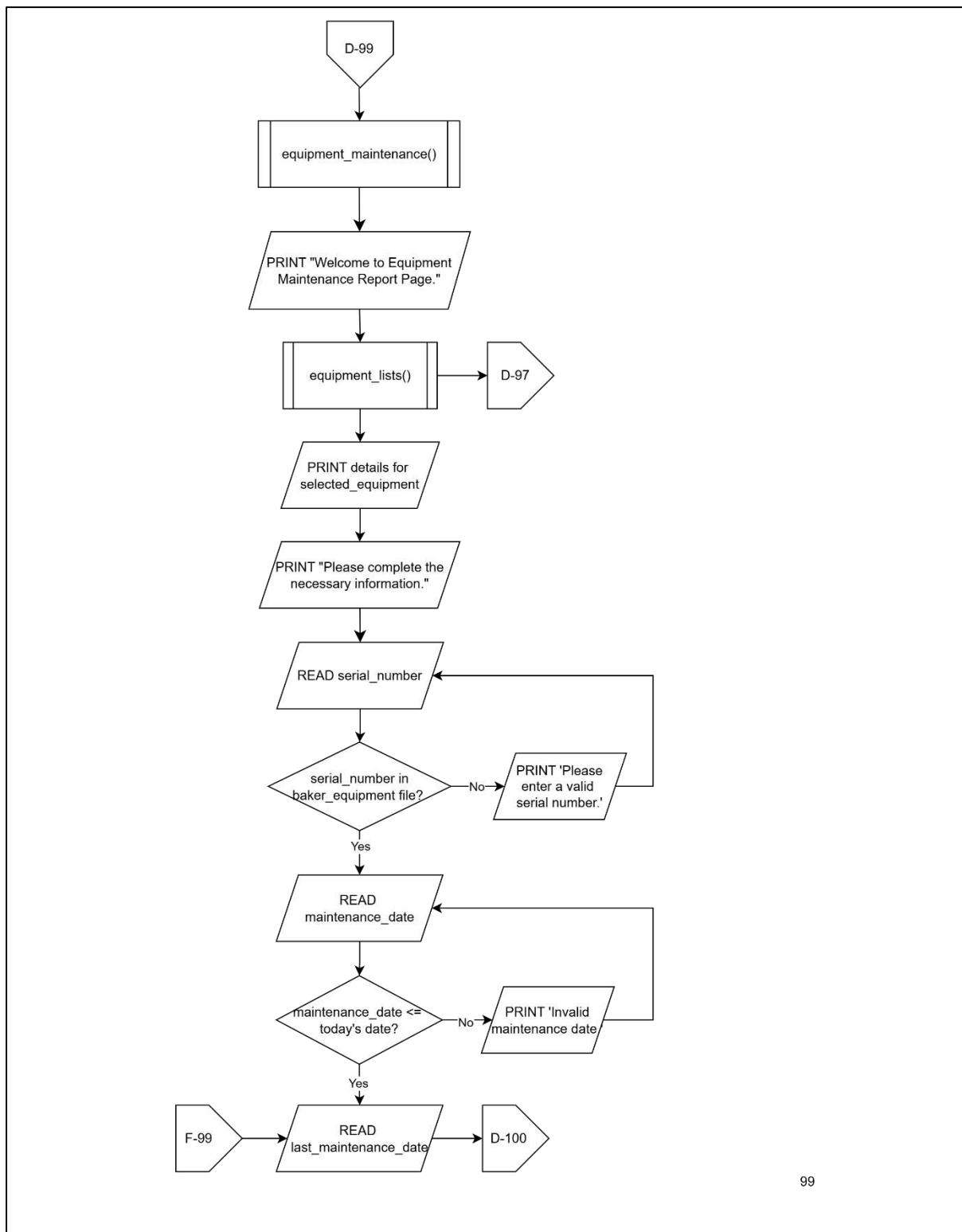
3.5.1 Equipment Management

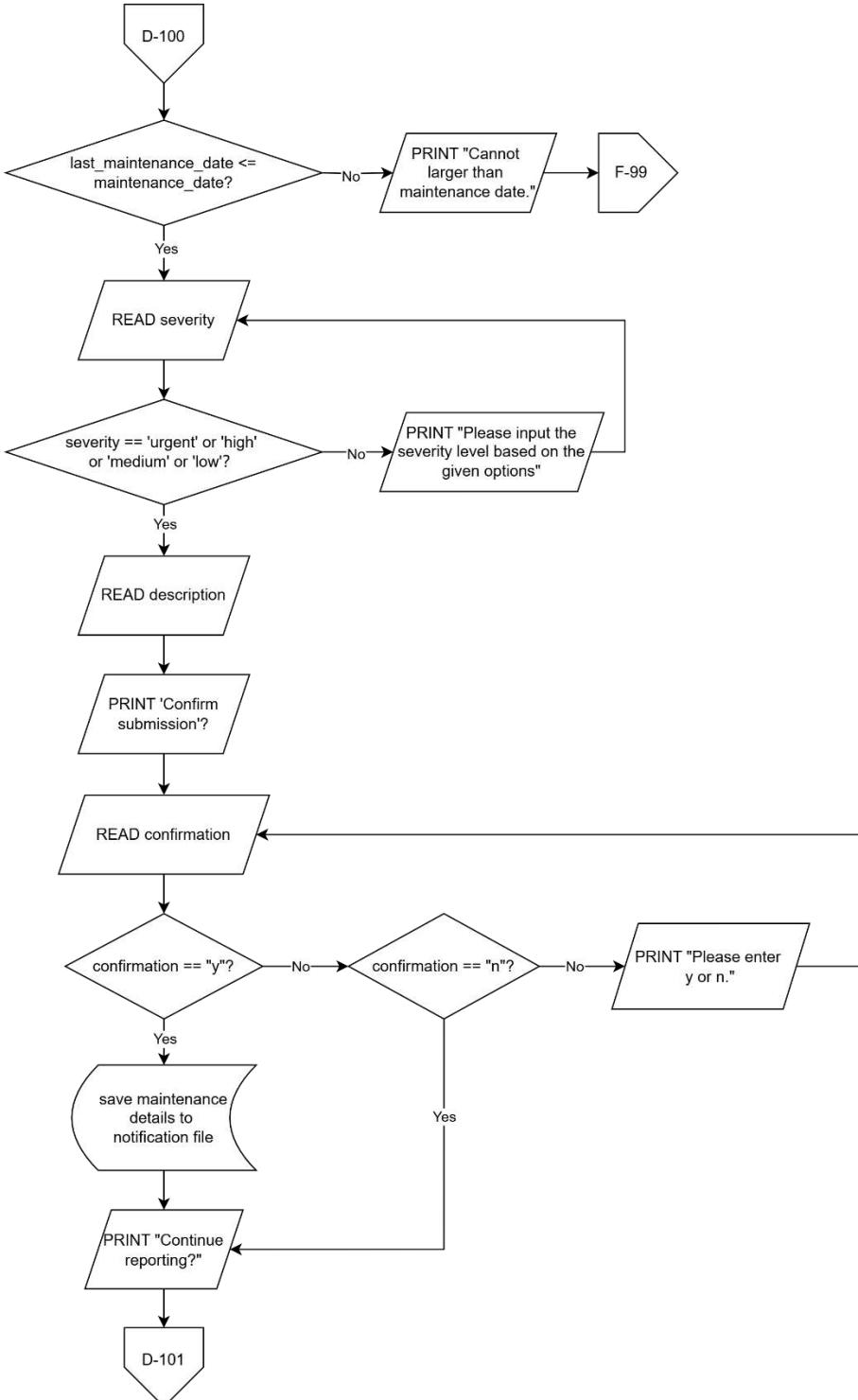




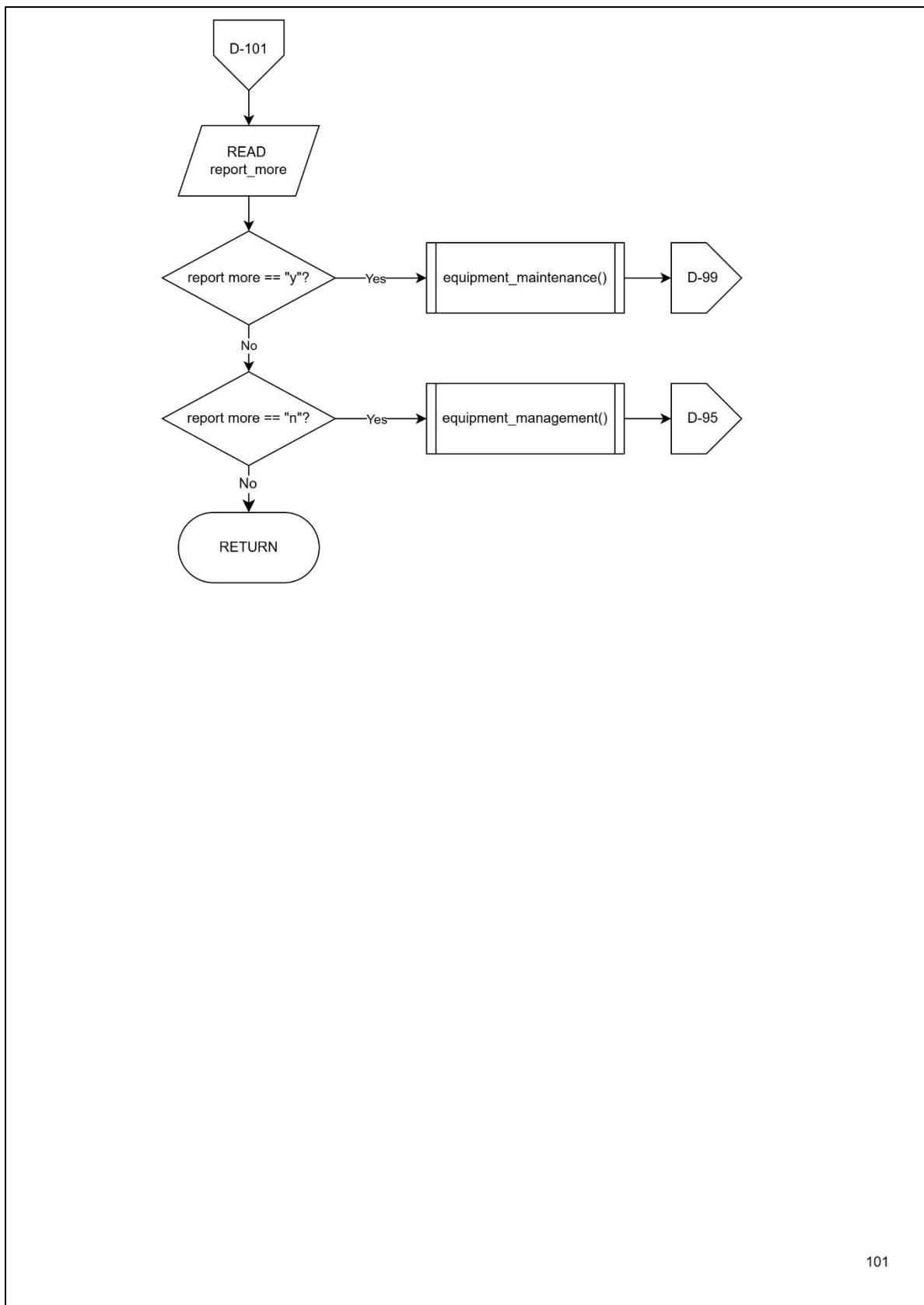








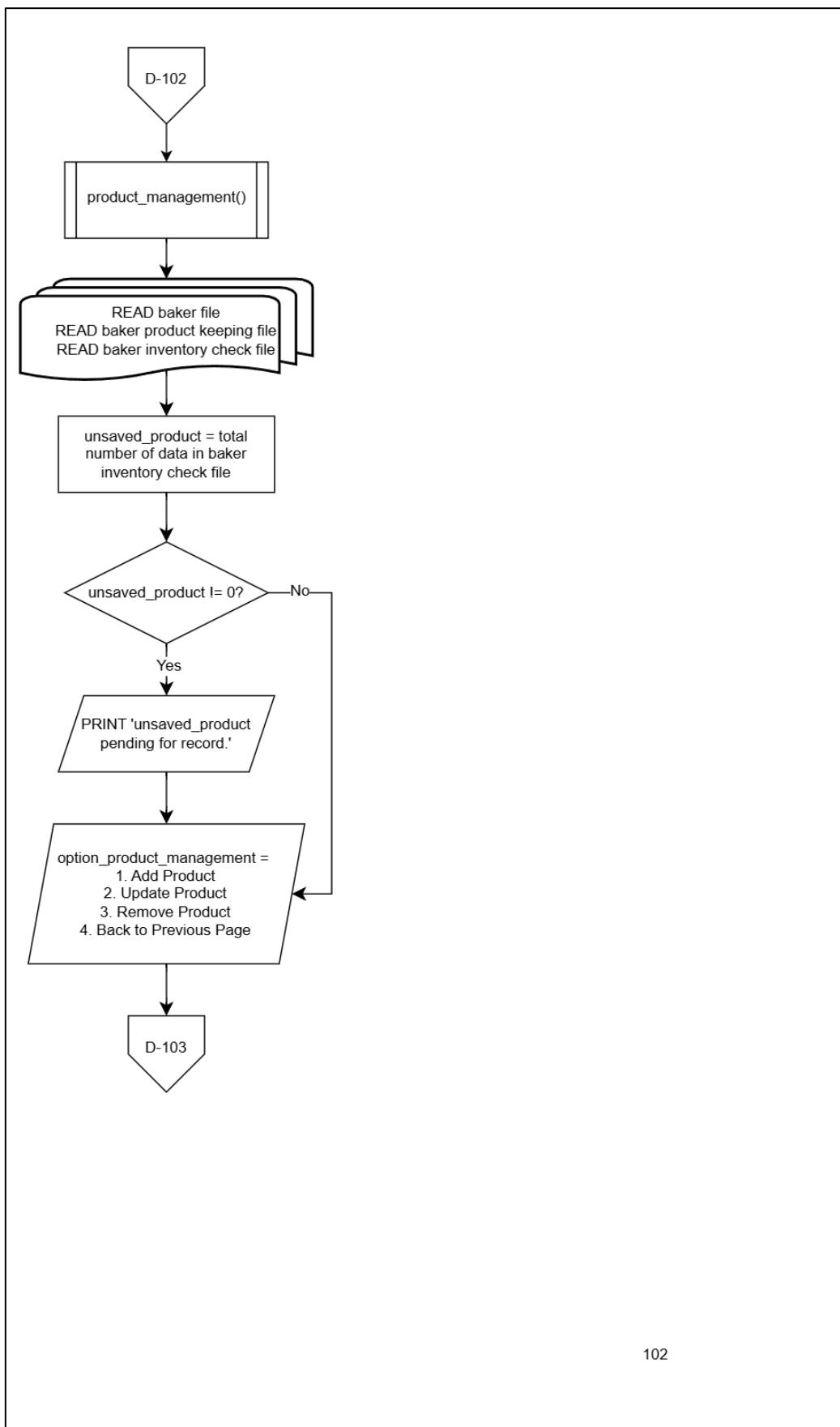
100

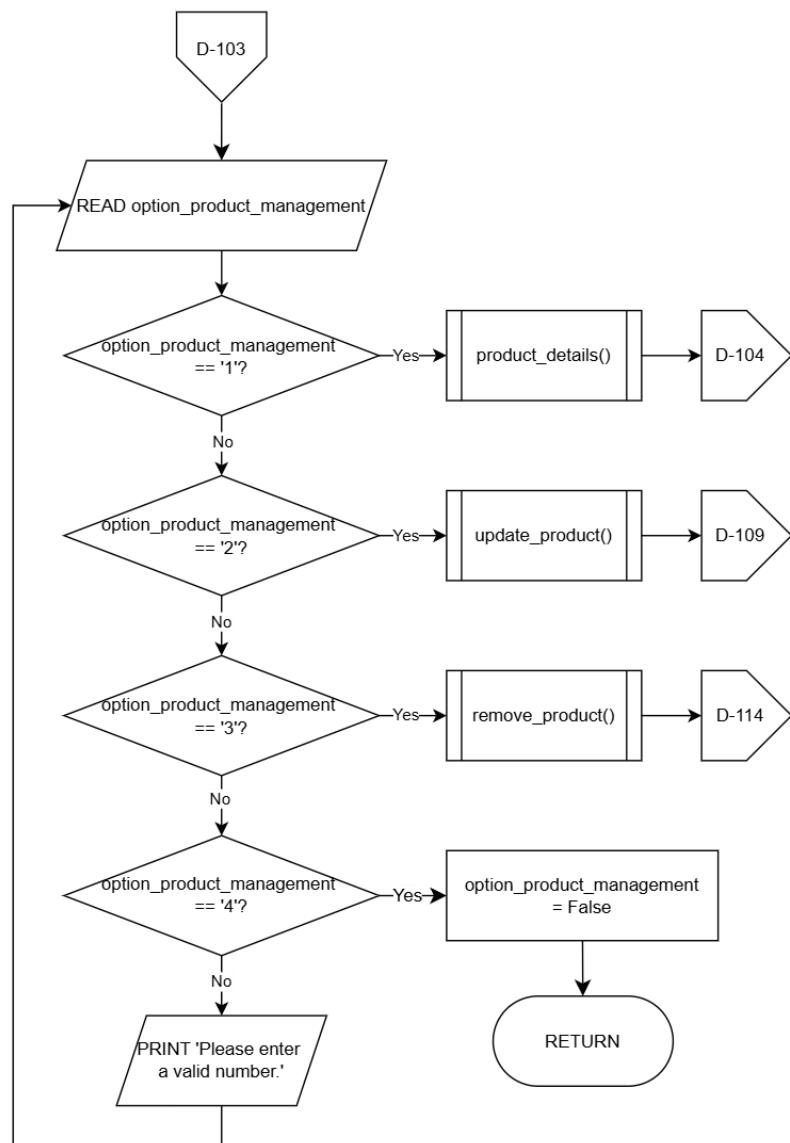


101

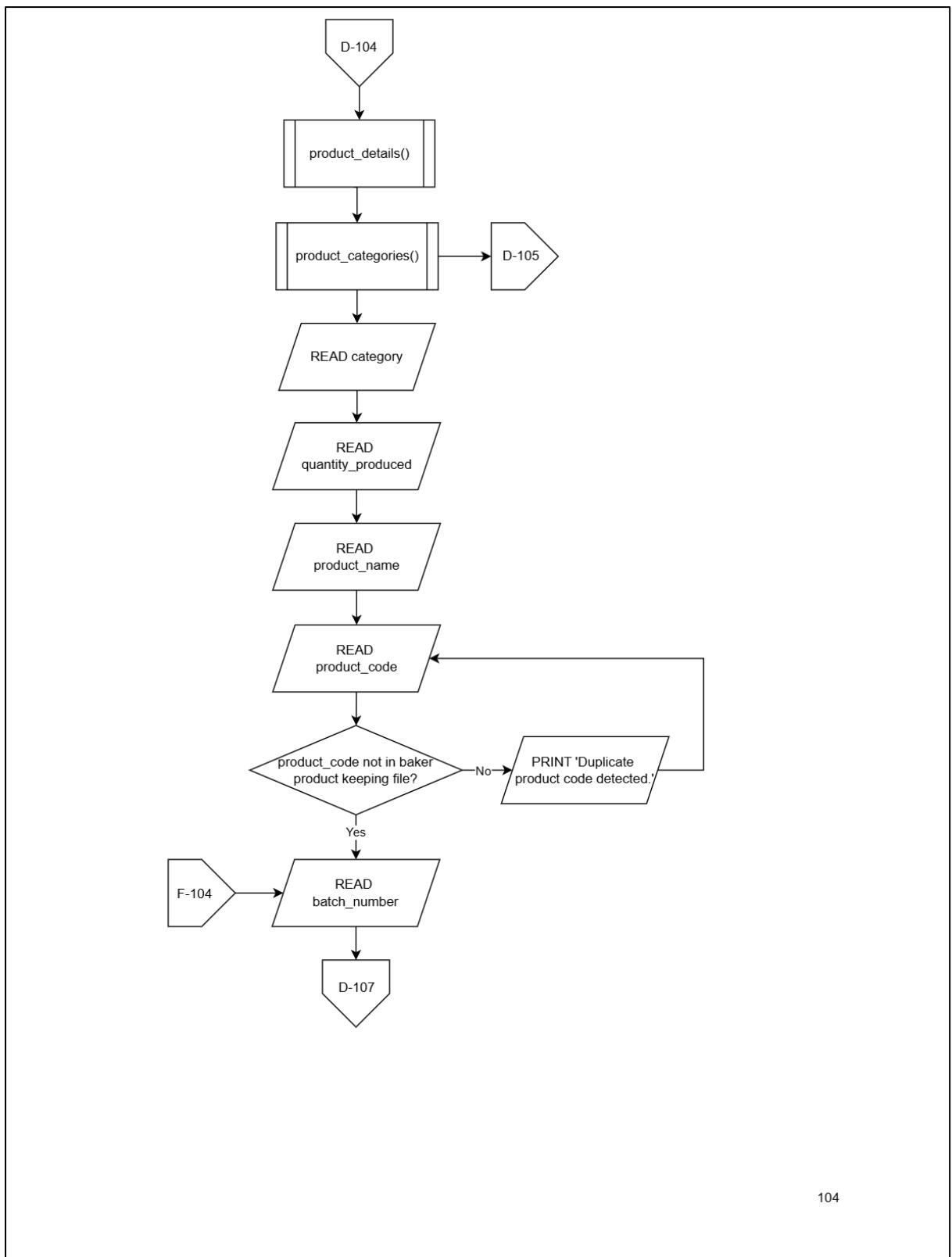
110

3.5.2 Product Keeping

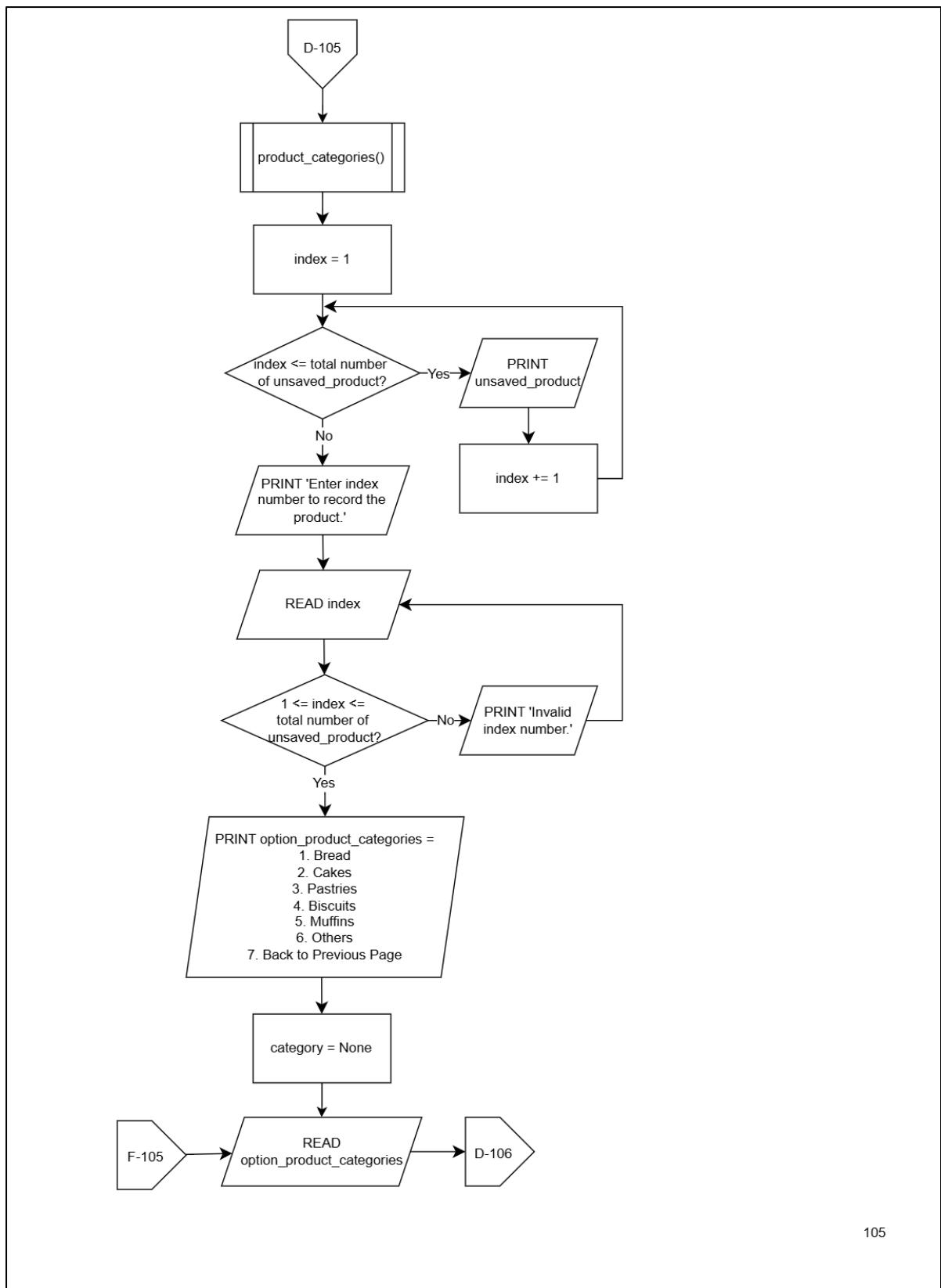




103

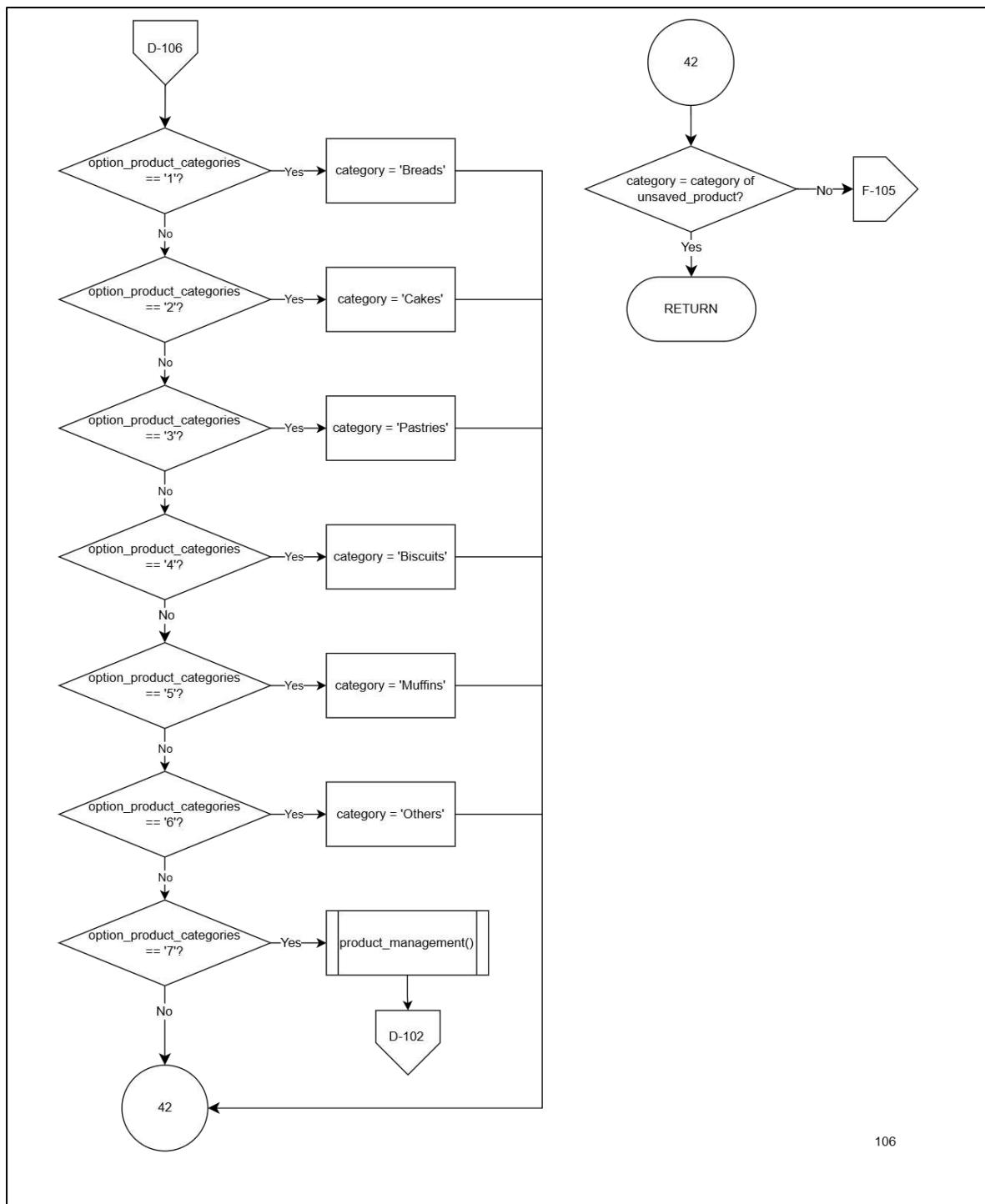


104

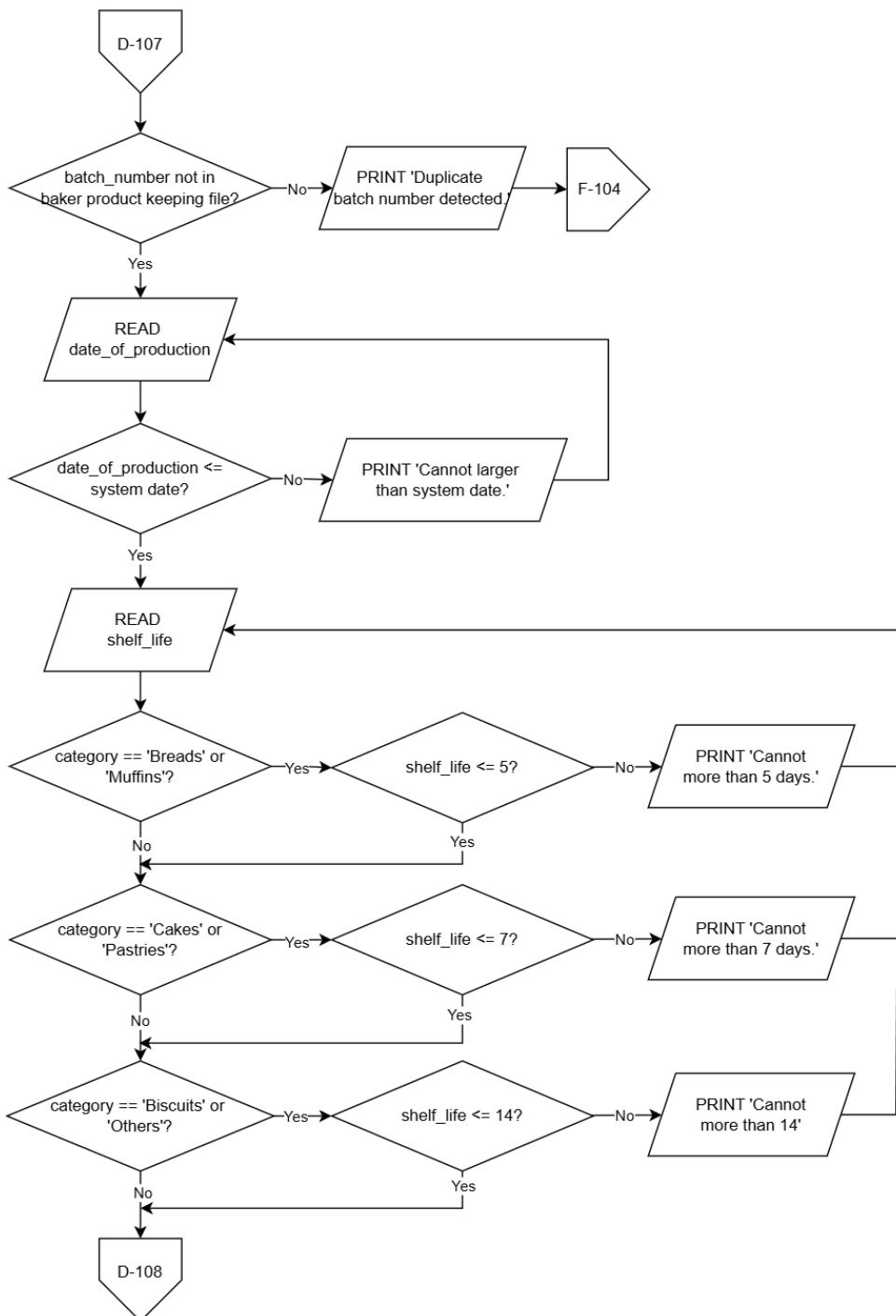


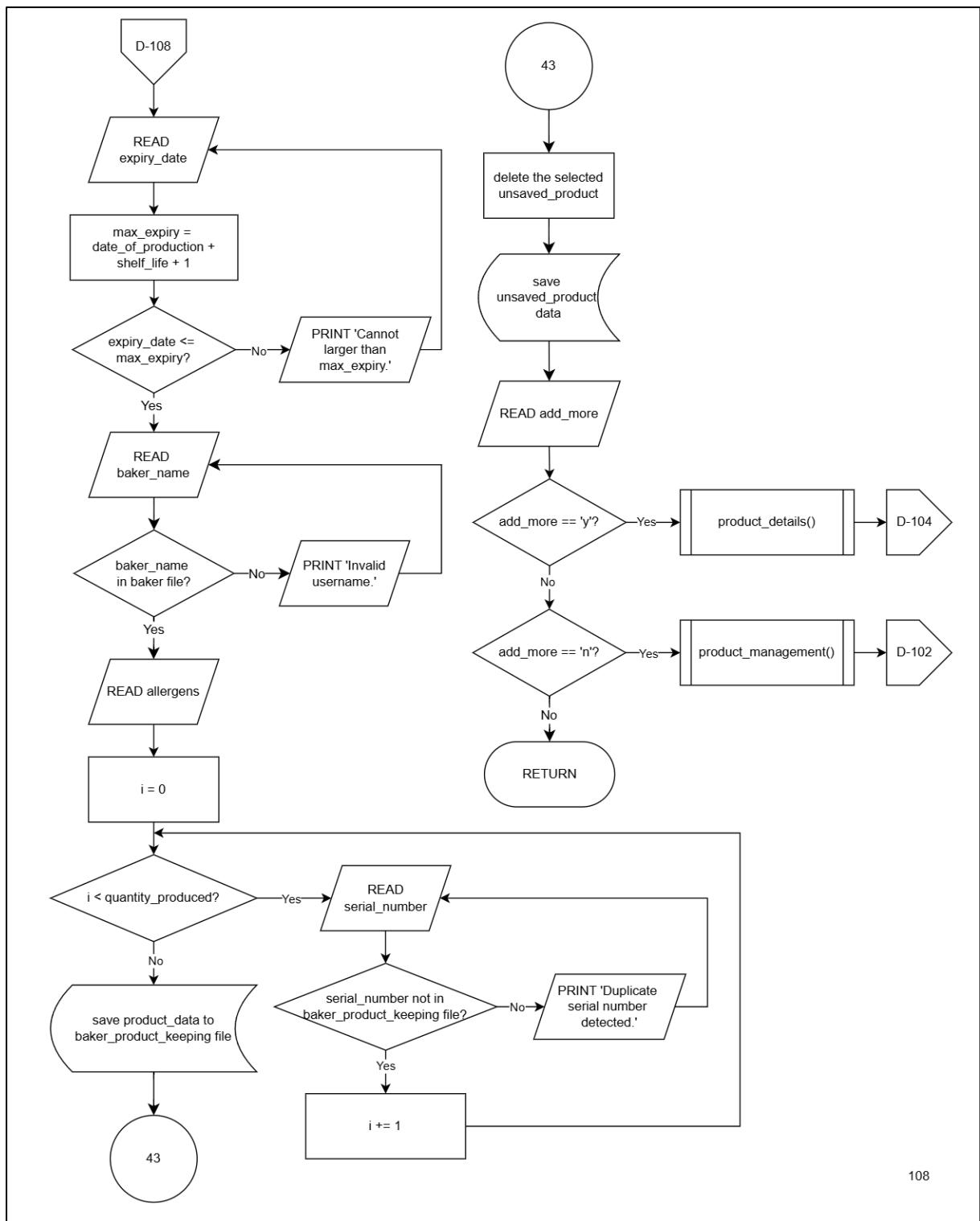
105

114

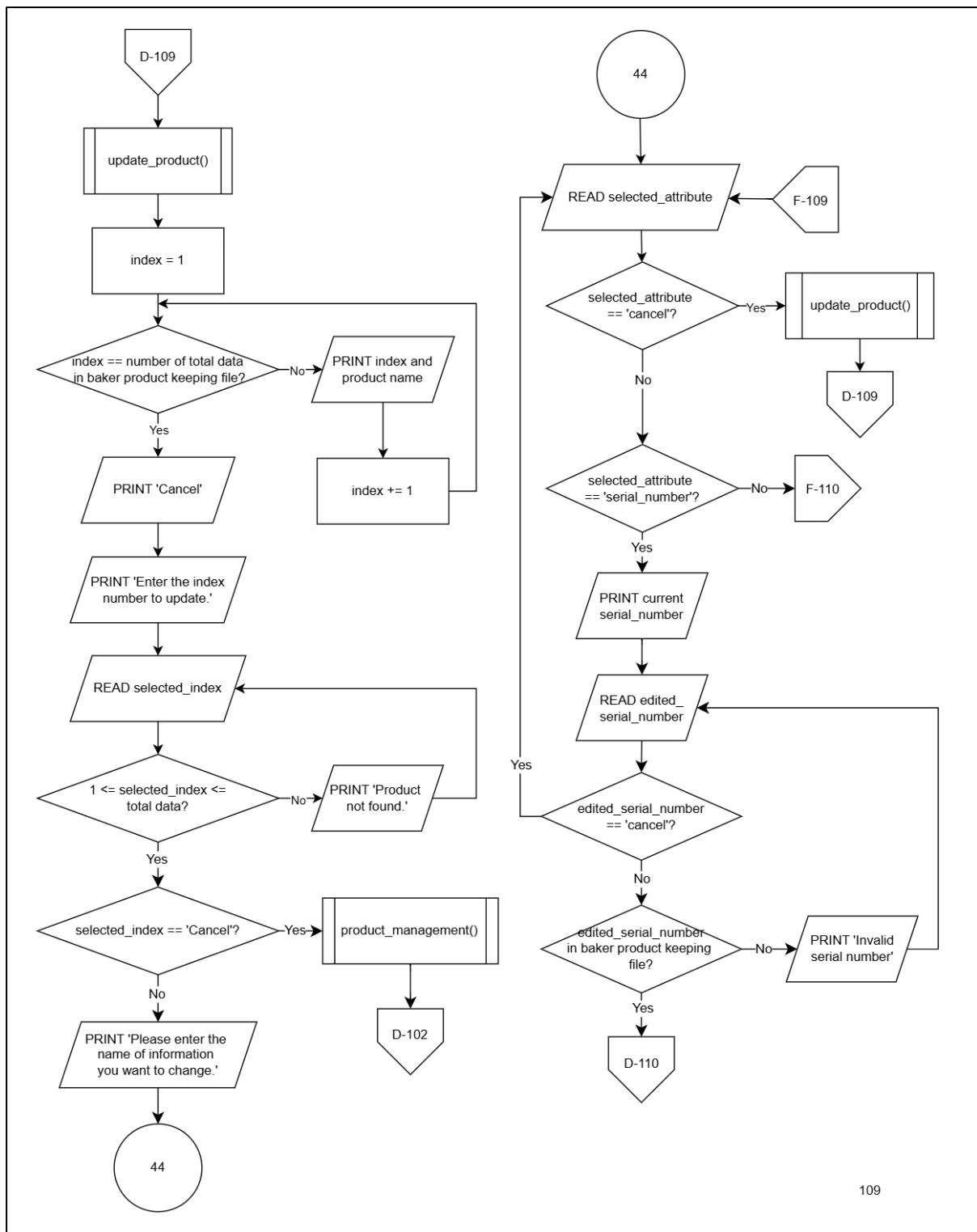


106

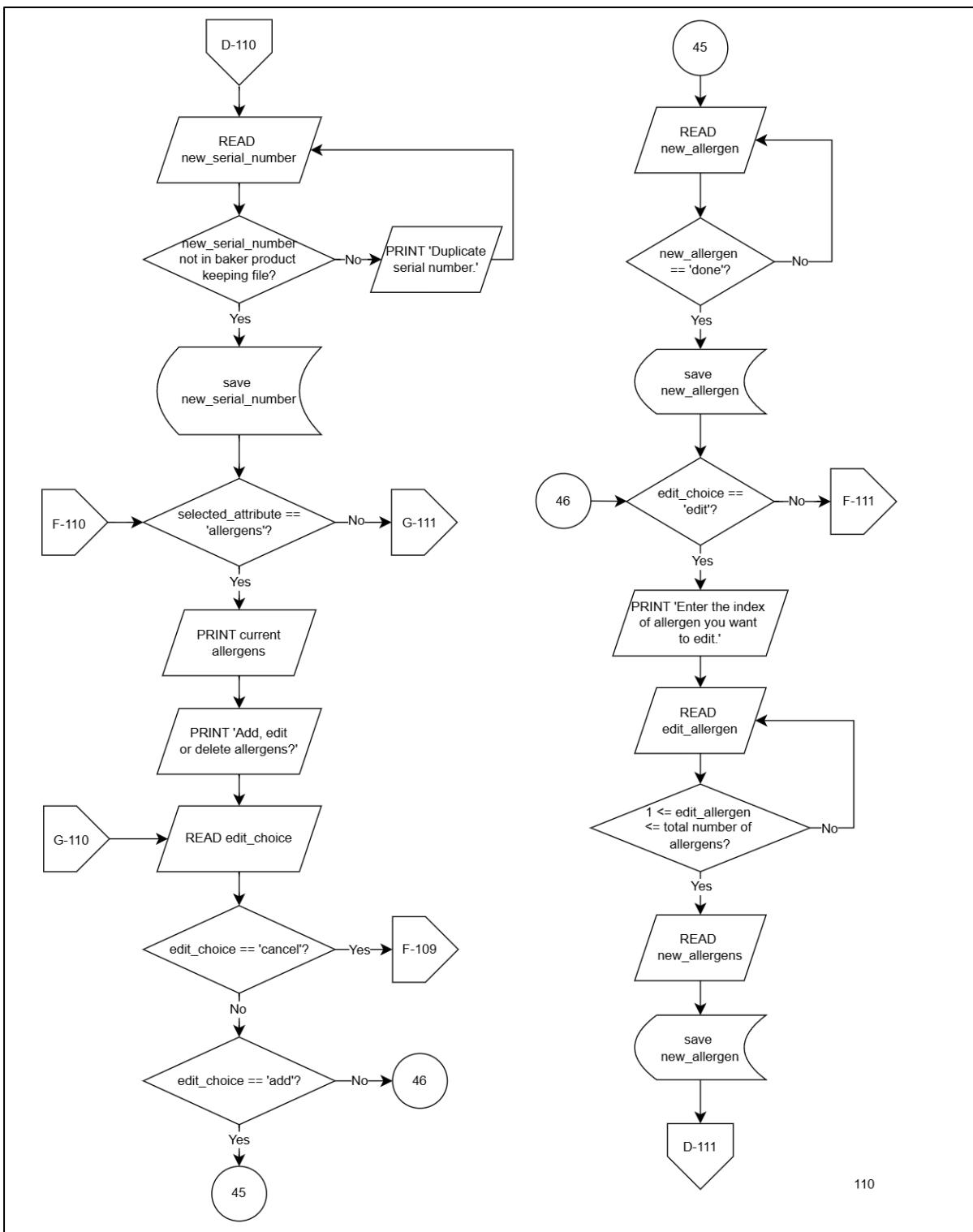


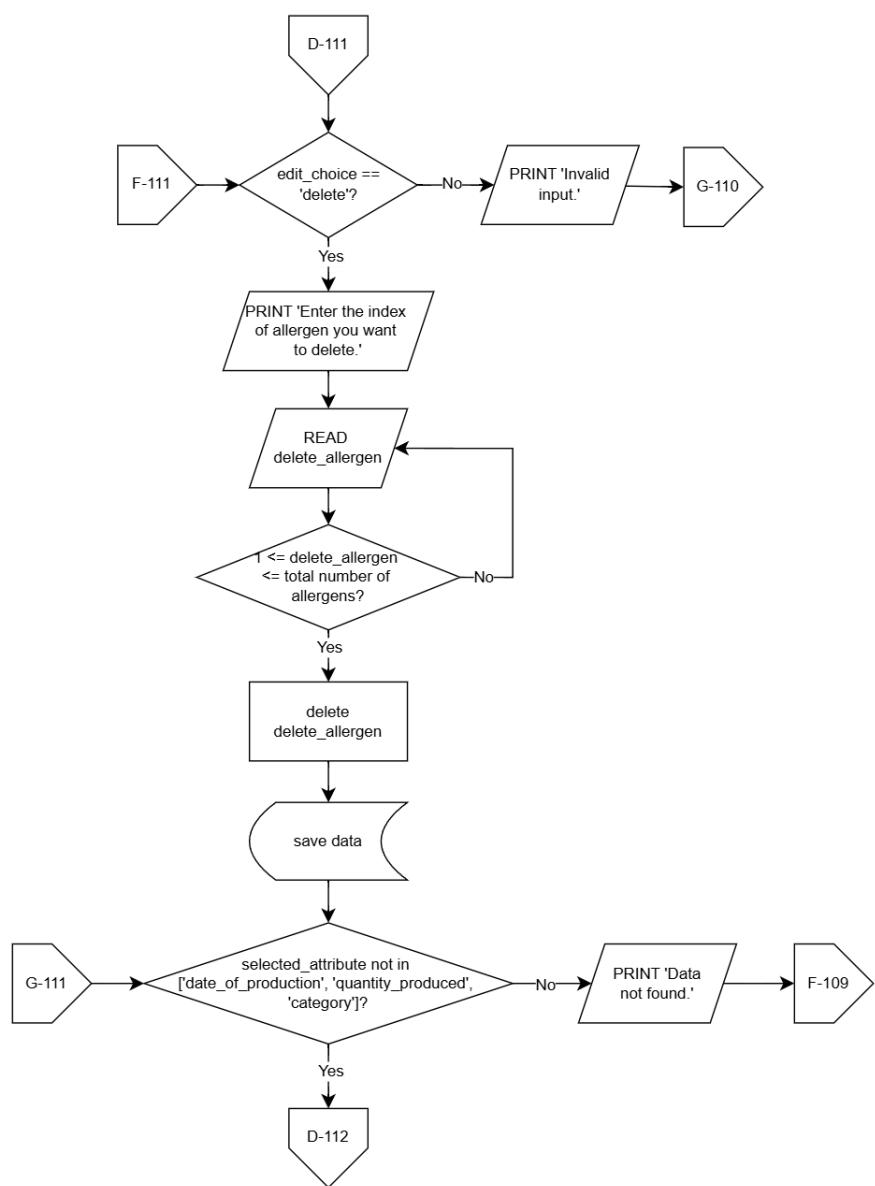


108

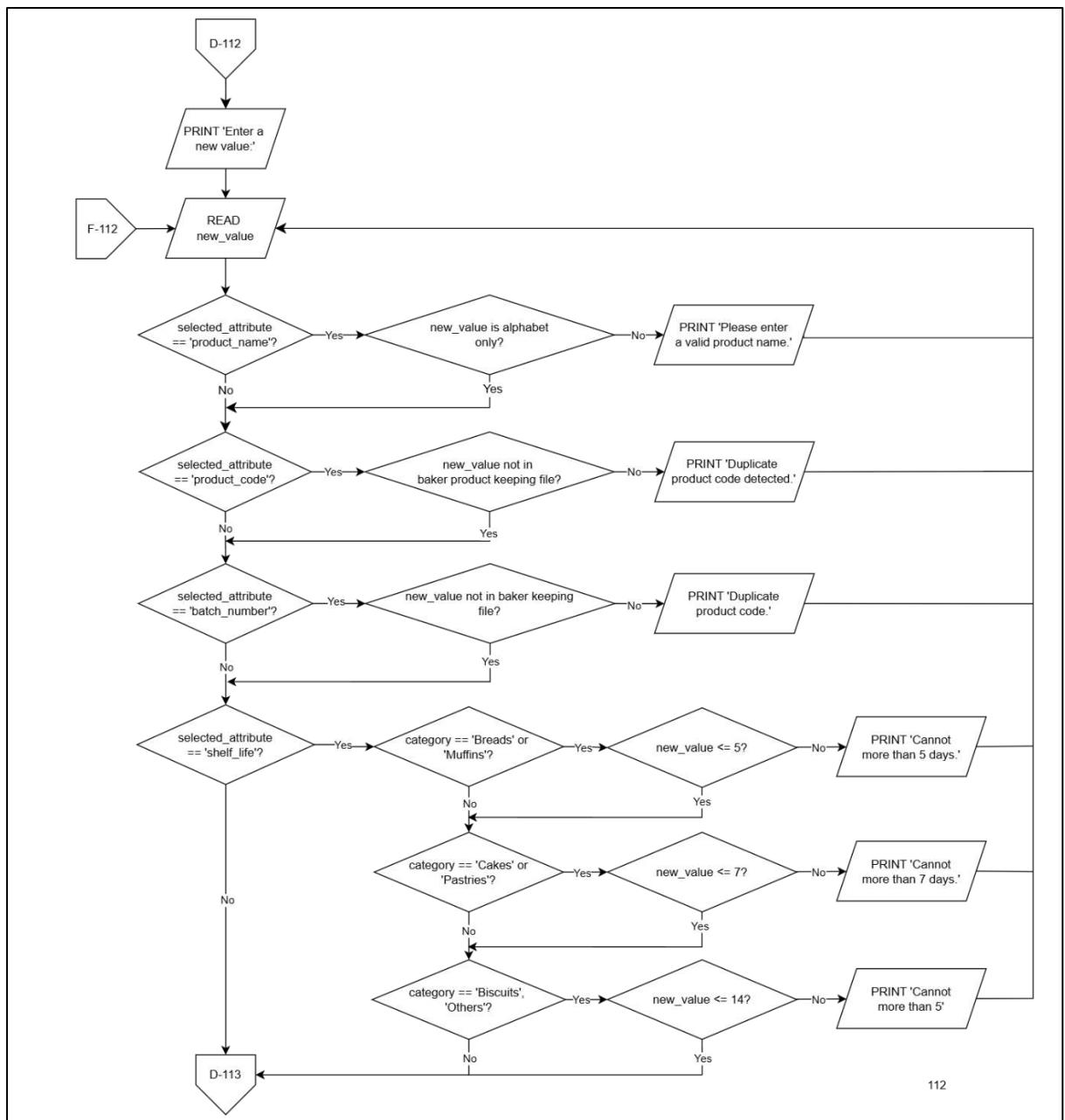


109

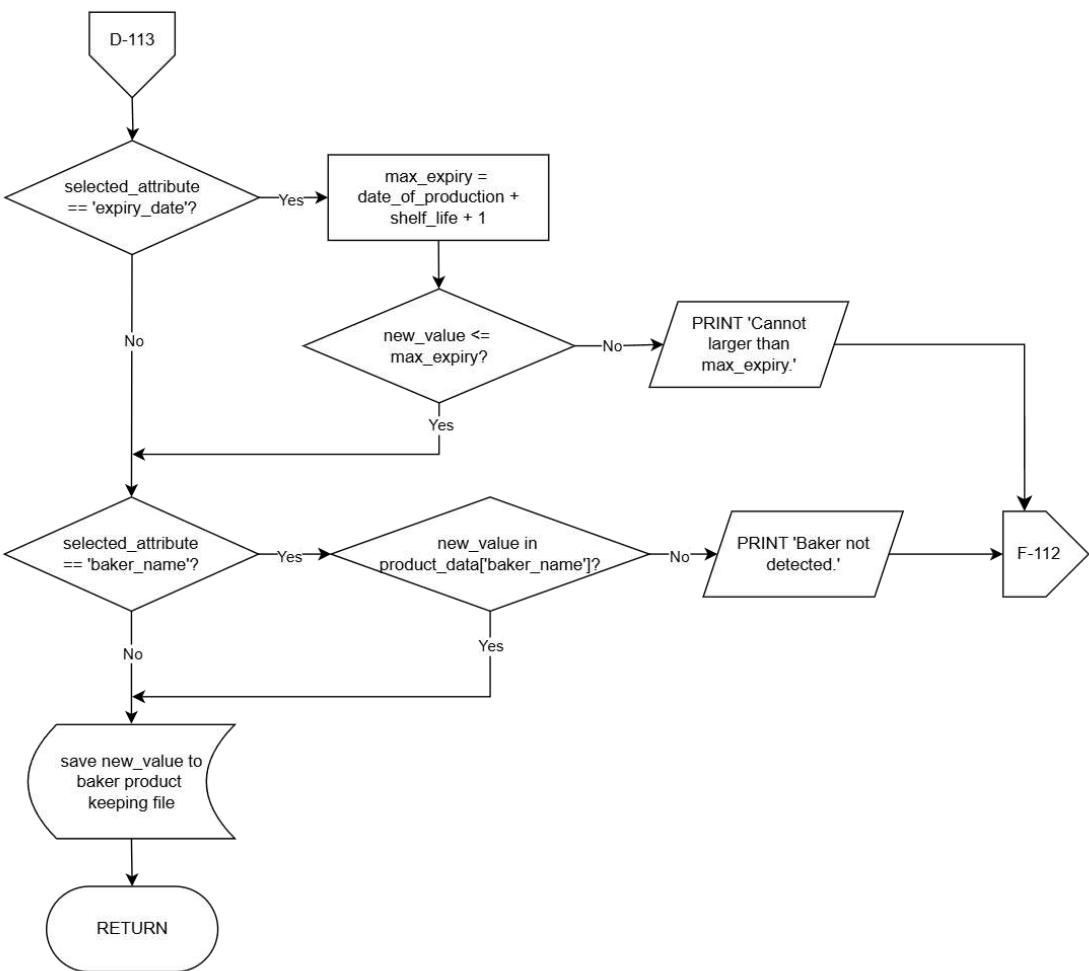




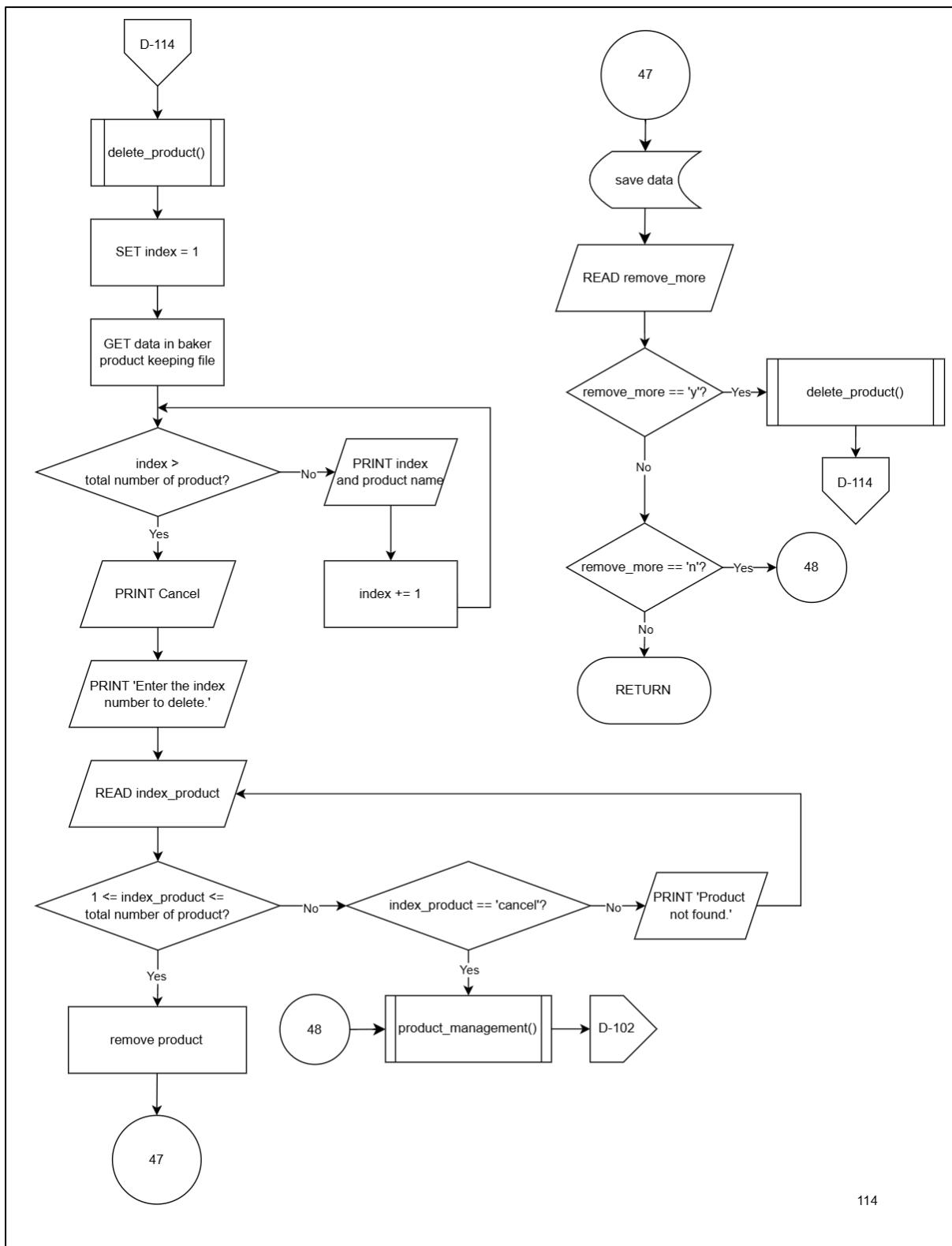
111



112

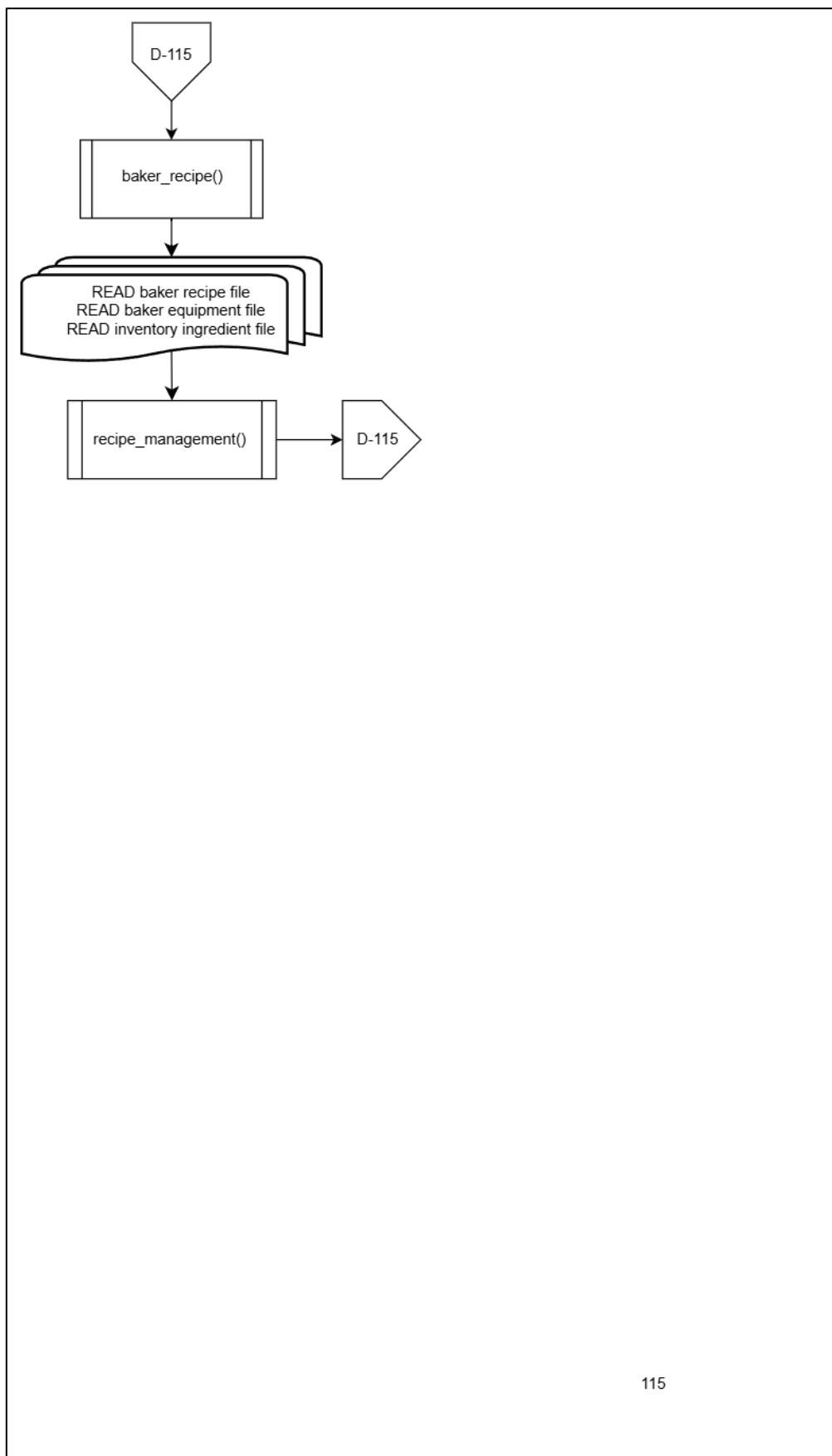


113

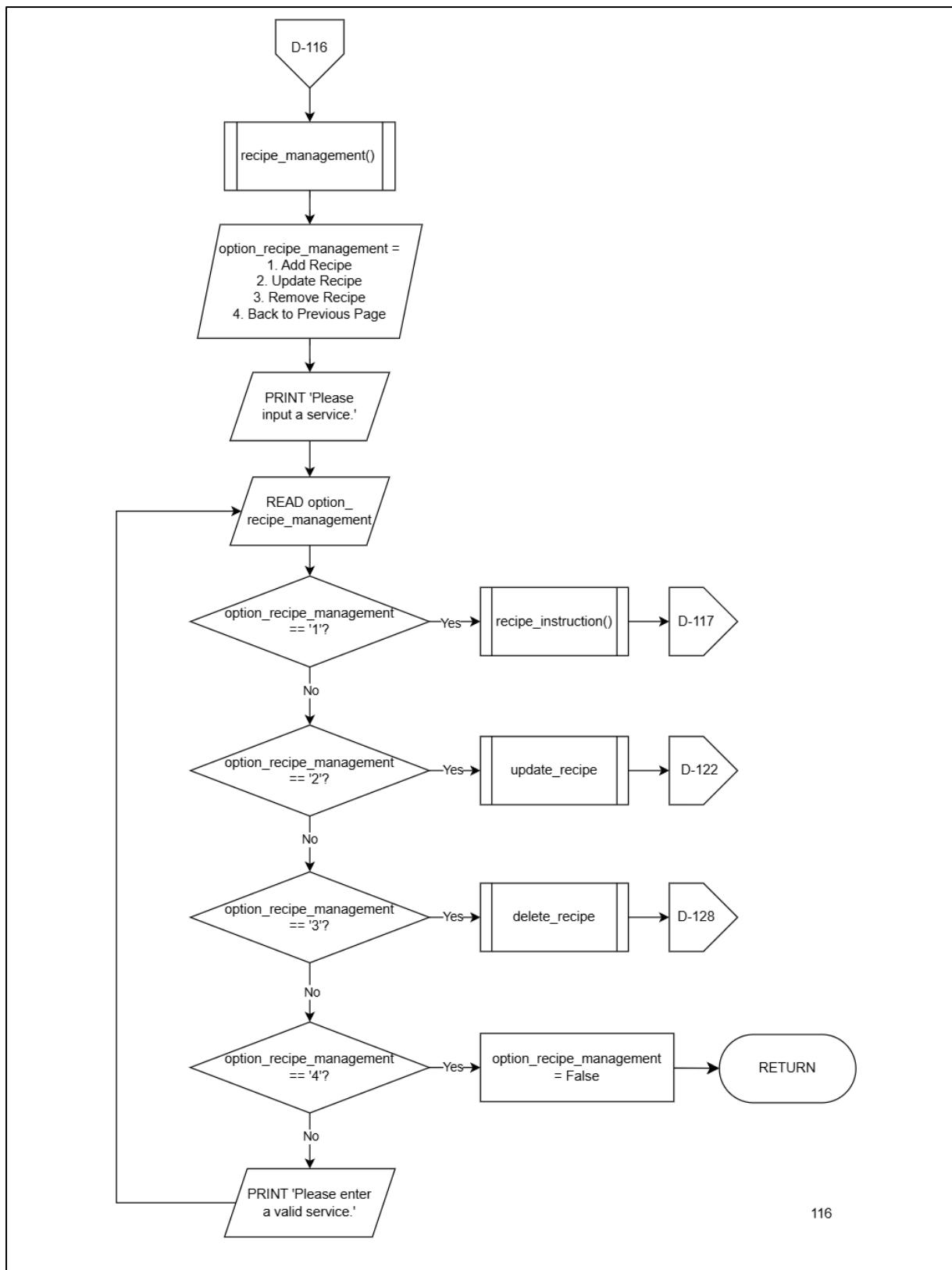


114

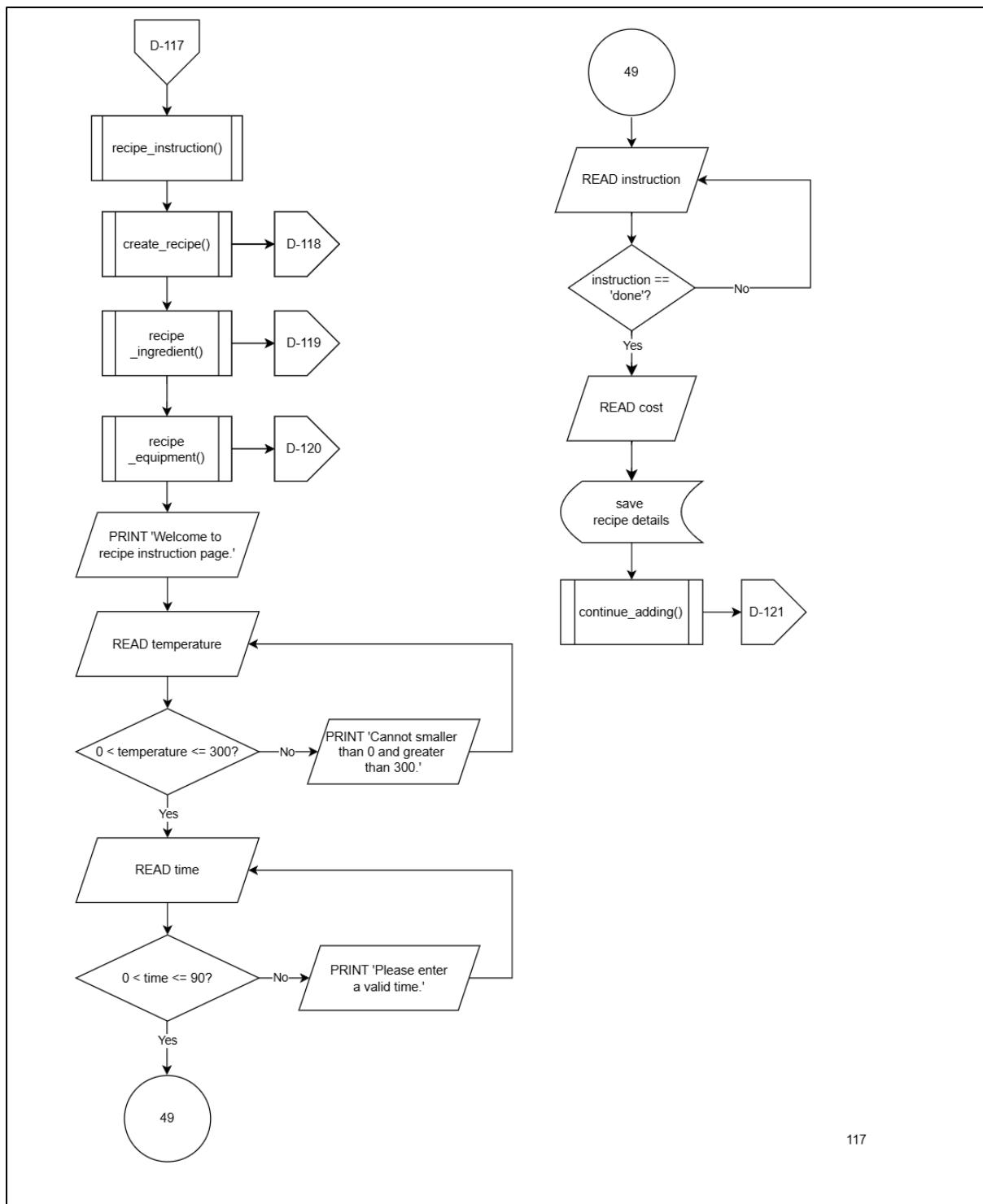
3.5.3 Recipe Management



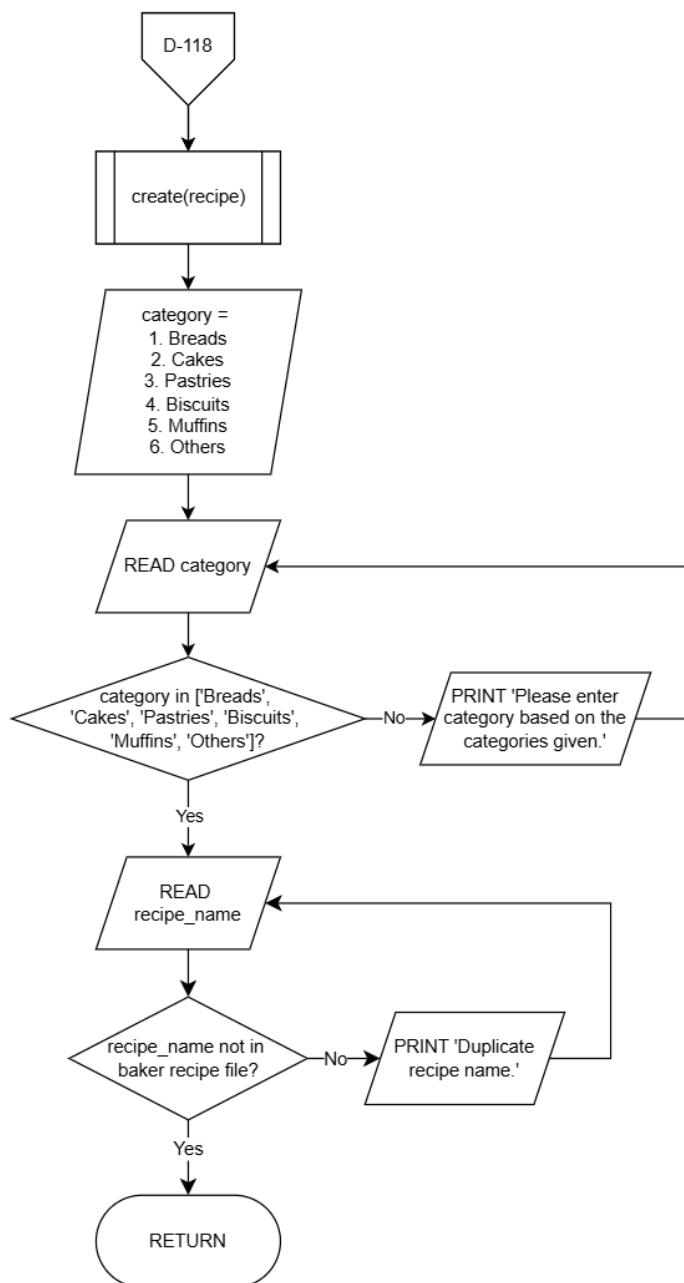
115

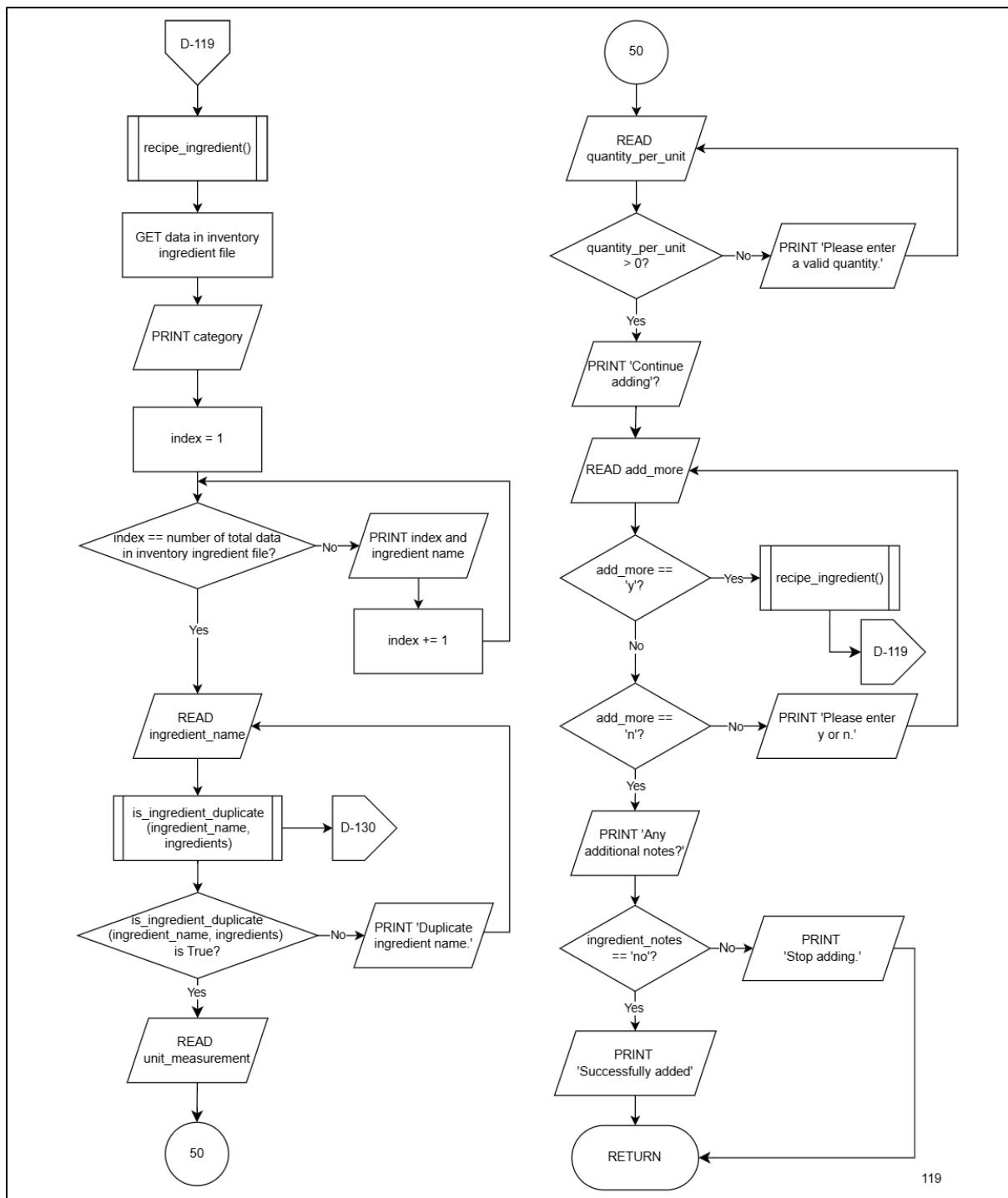


116

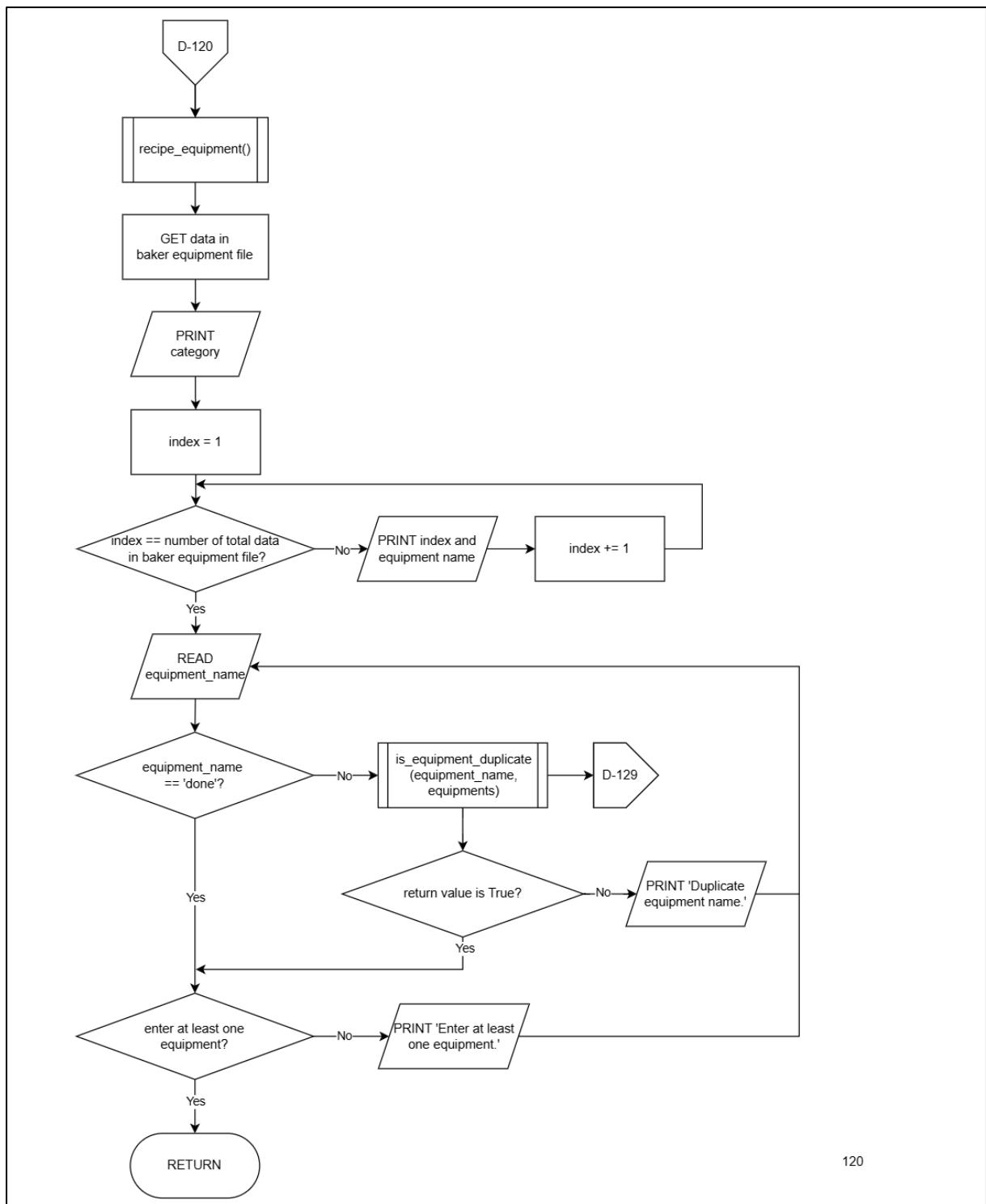


117

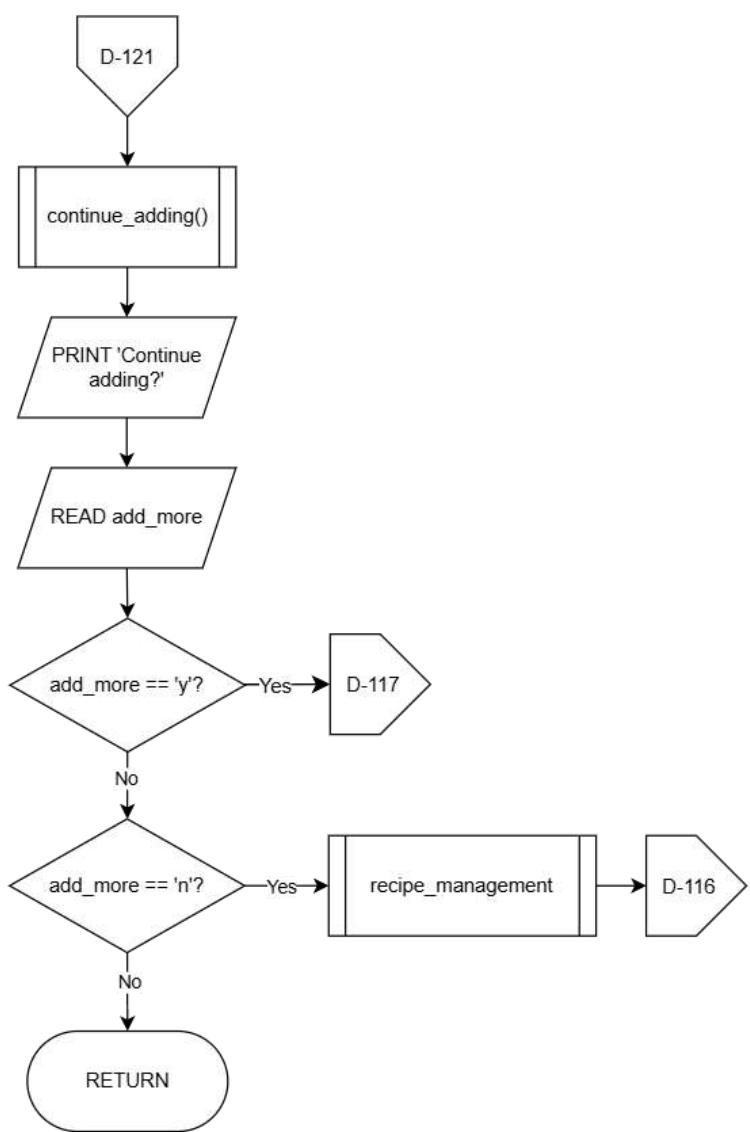


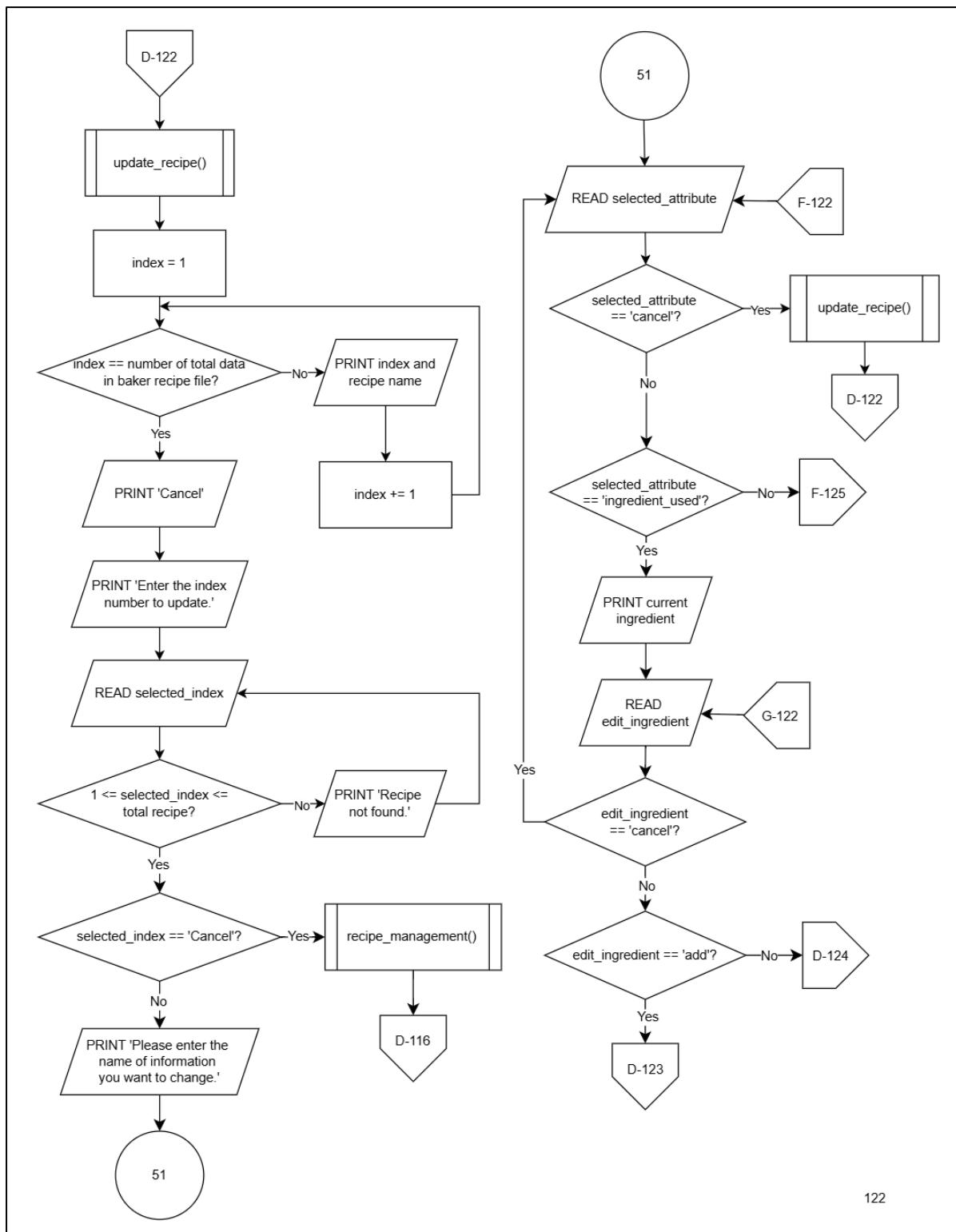


119

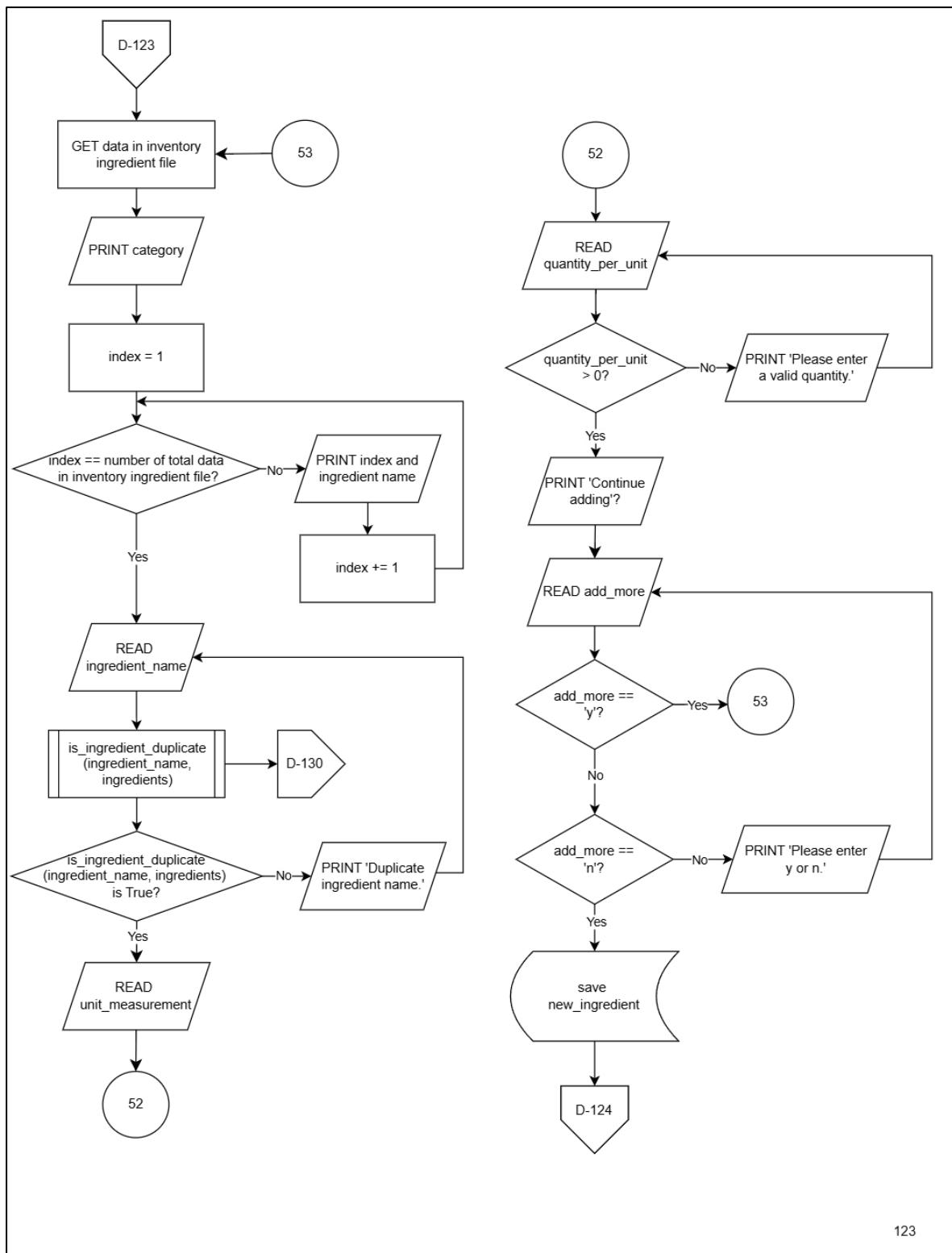


120

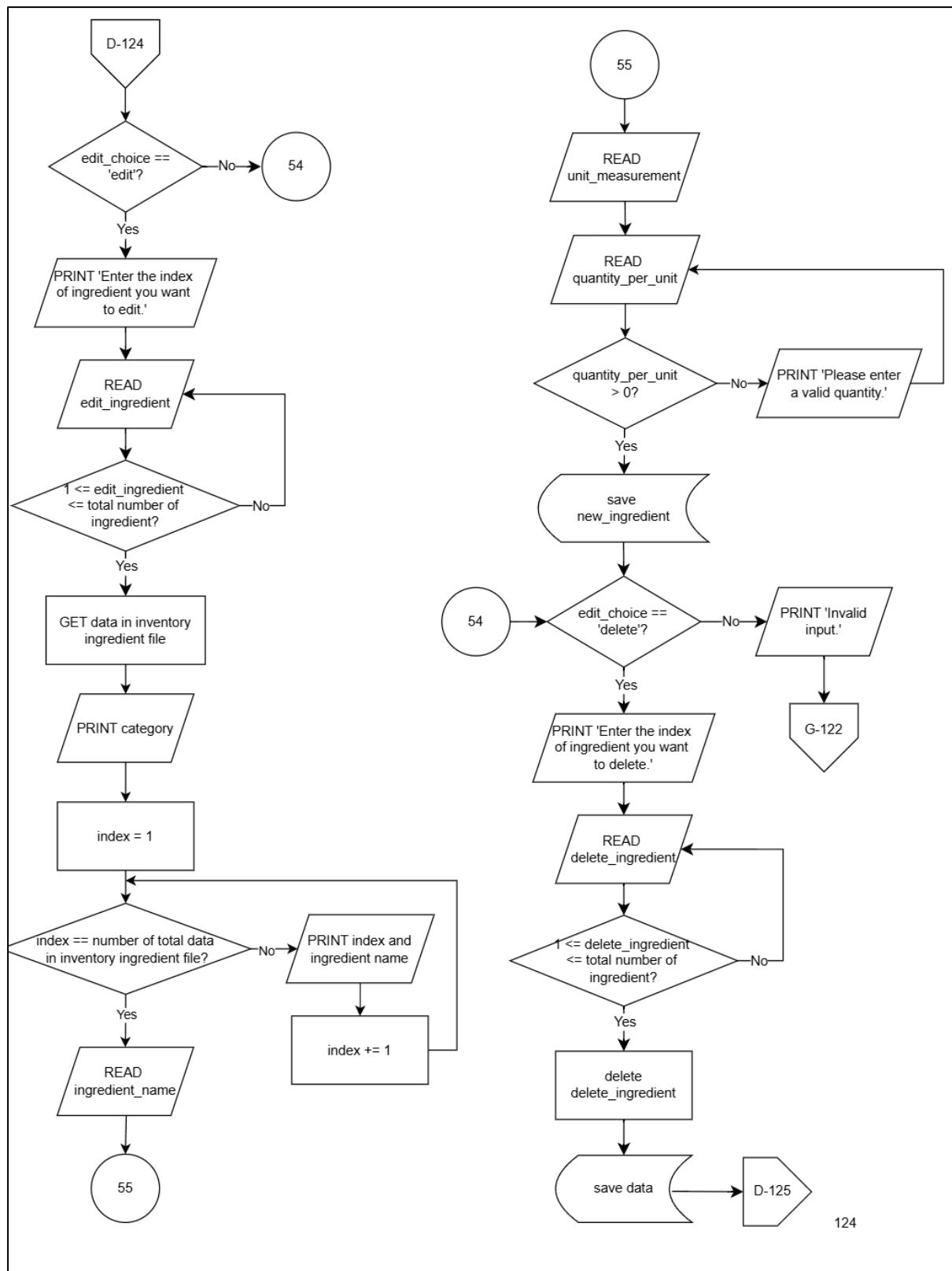


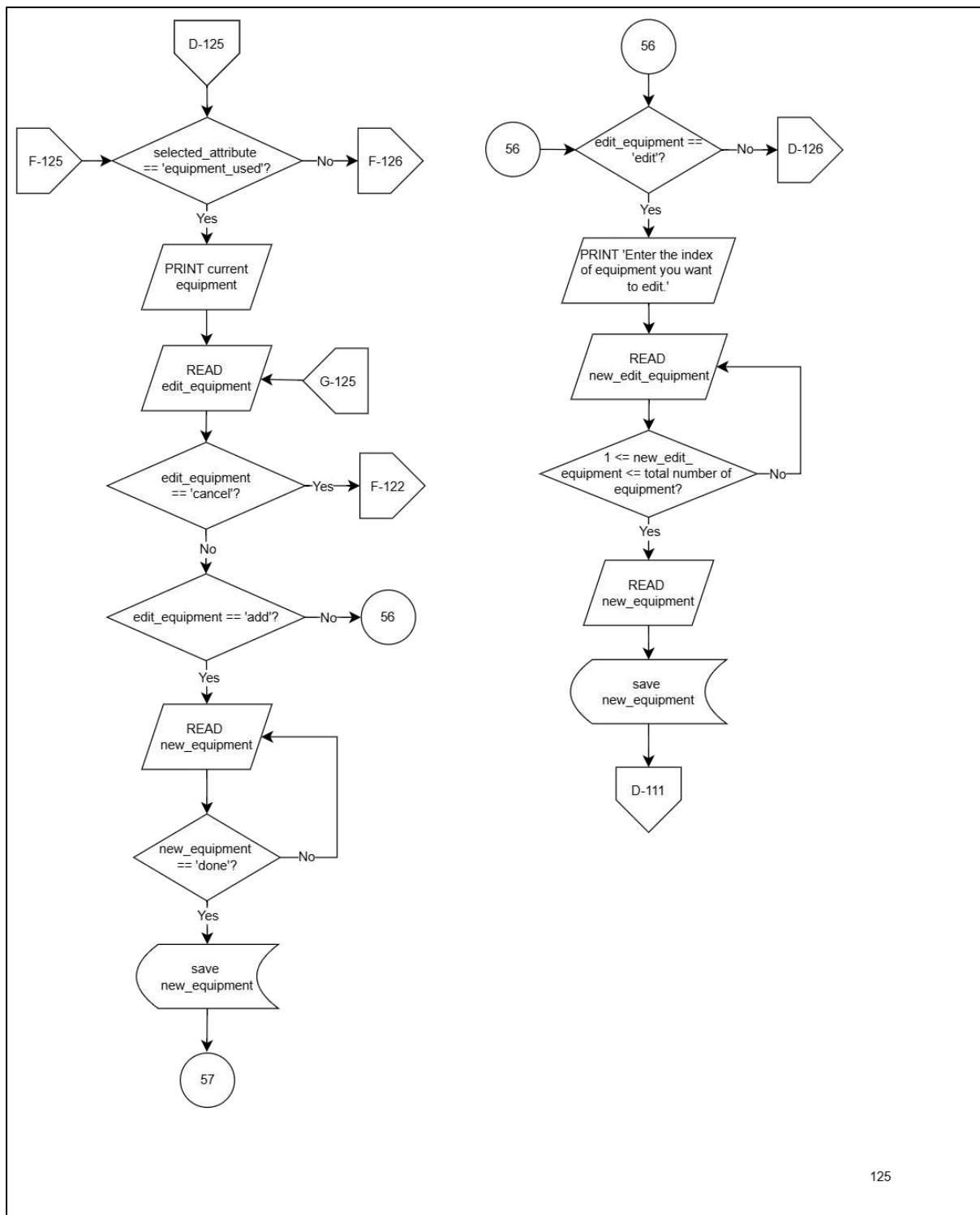


122

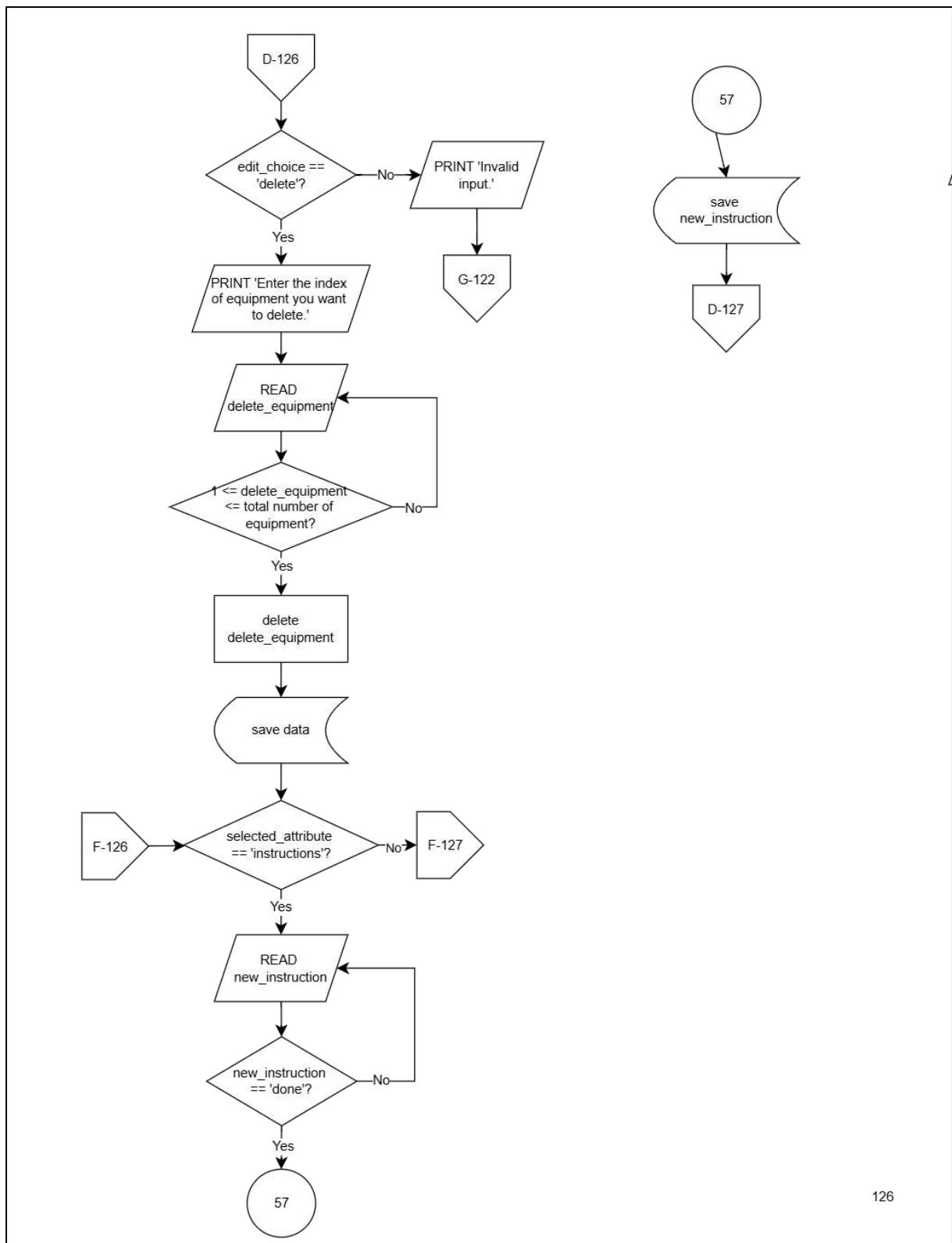


123



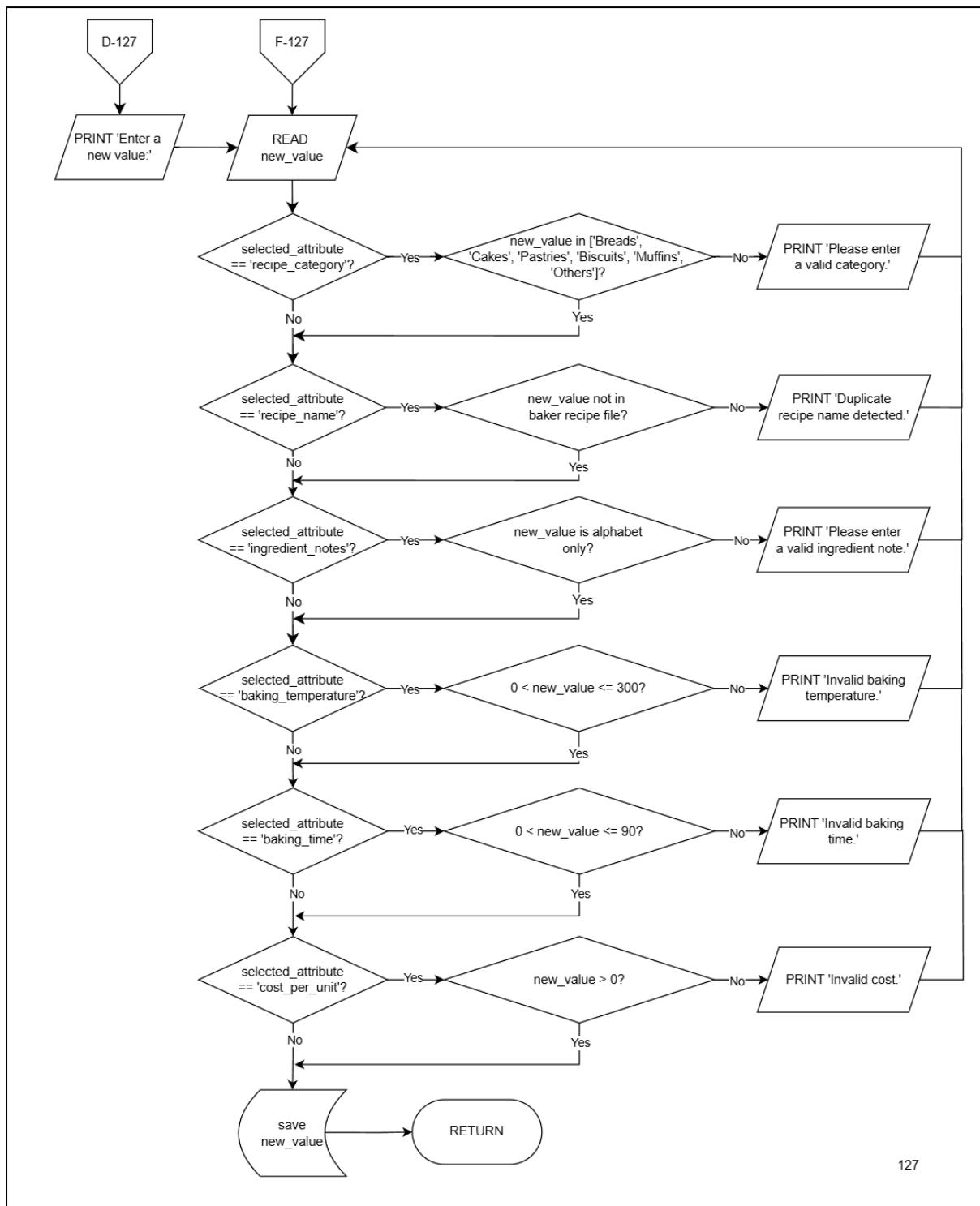


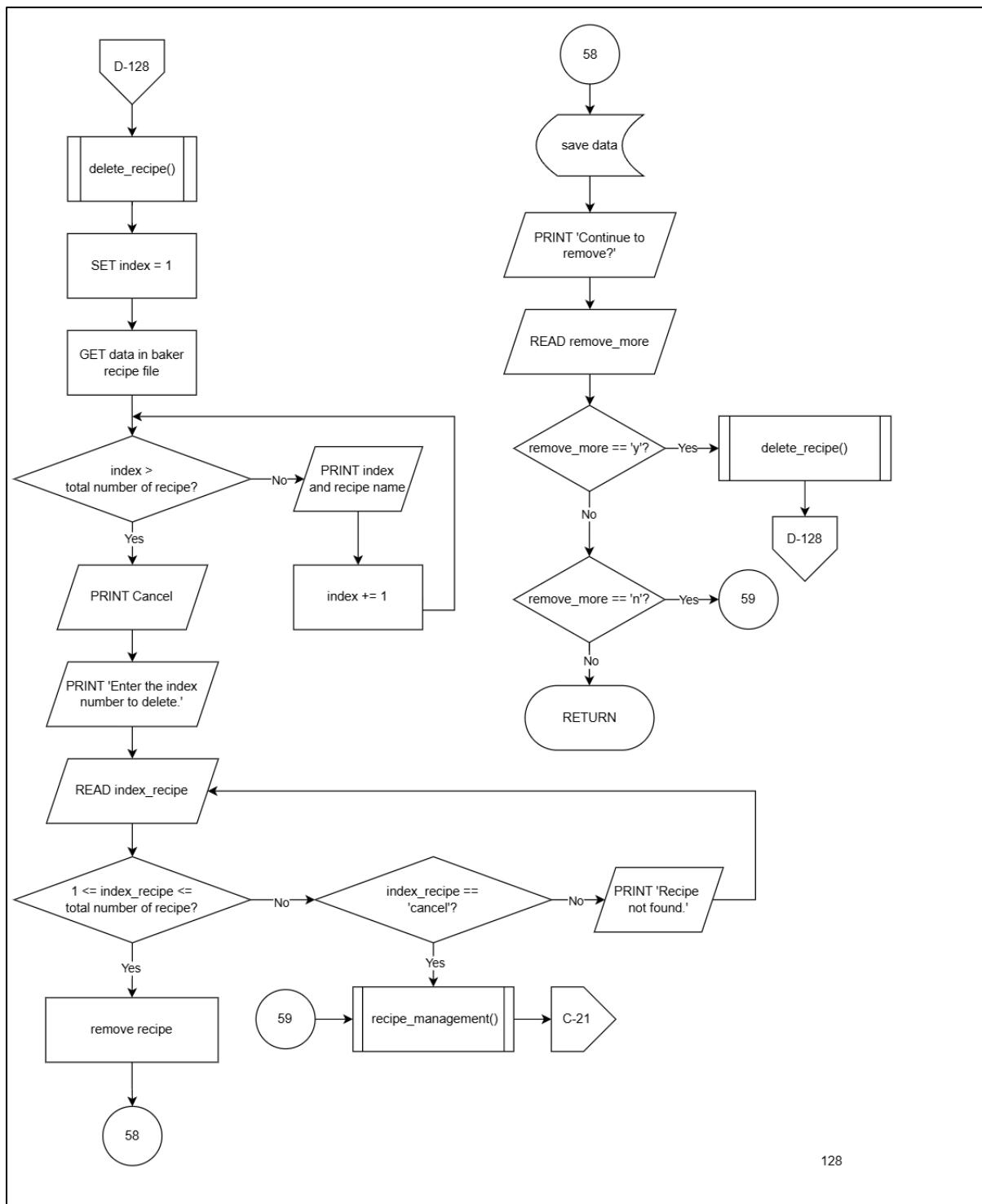
125



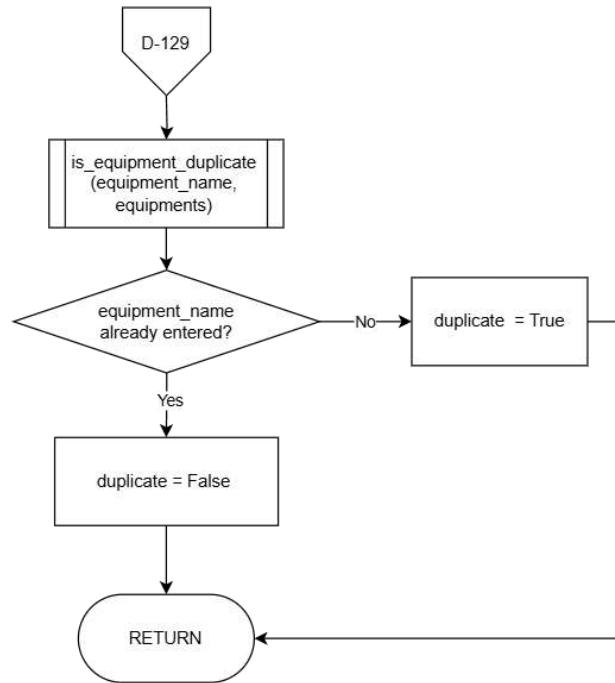
126

135

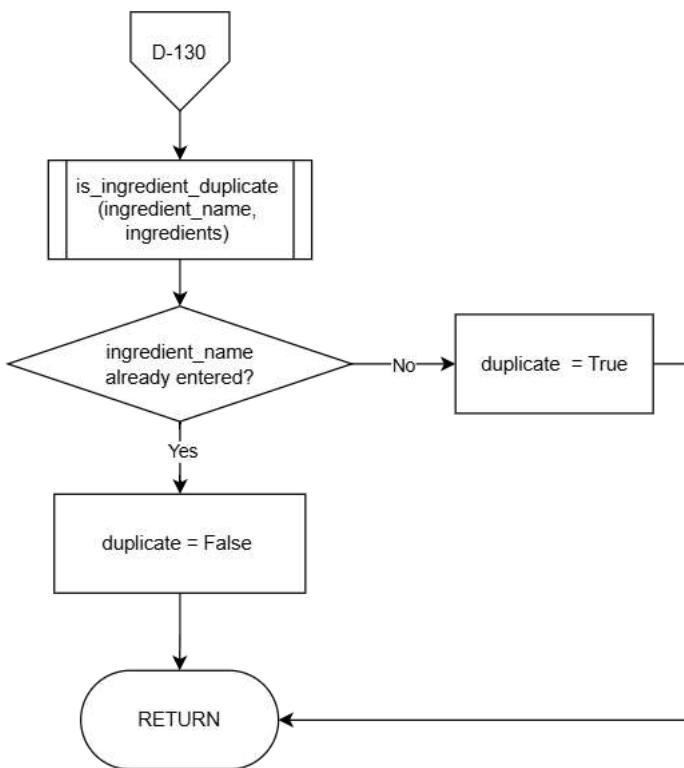




128

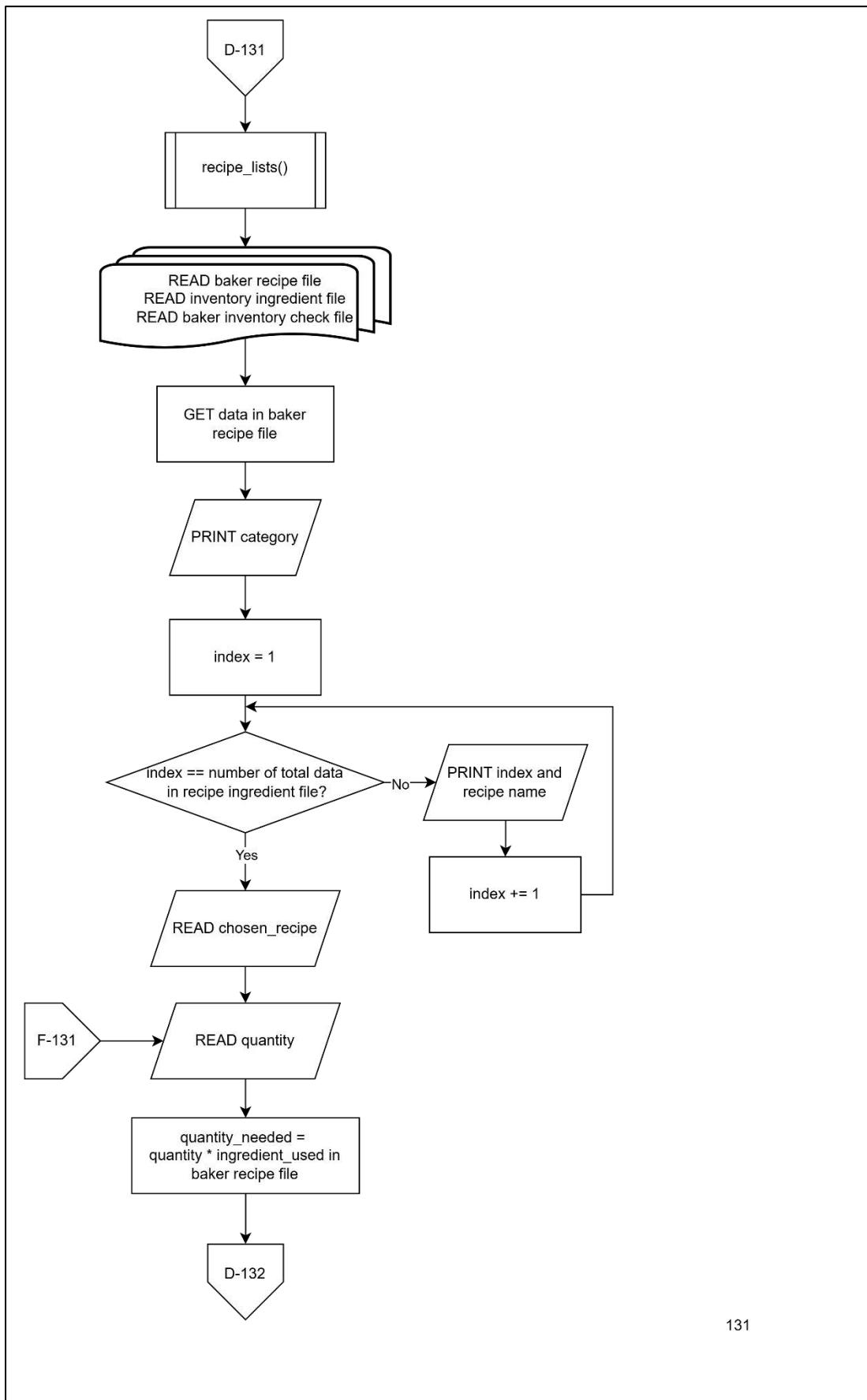


129

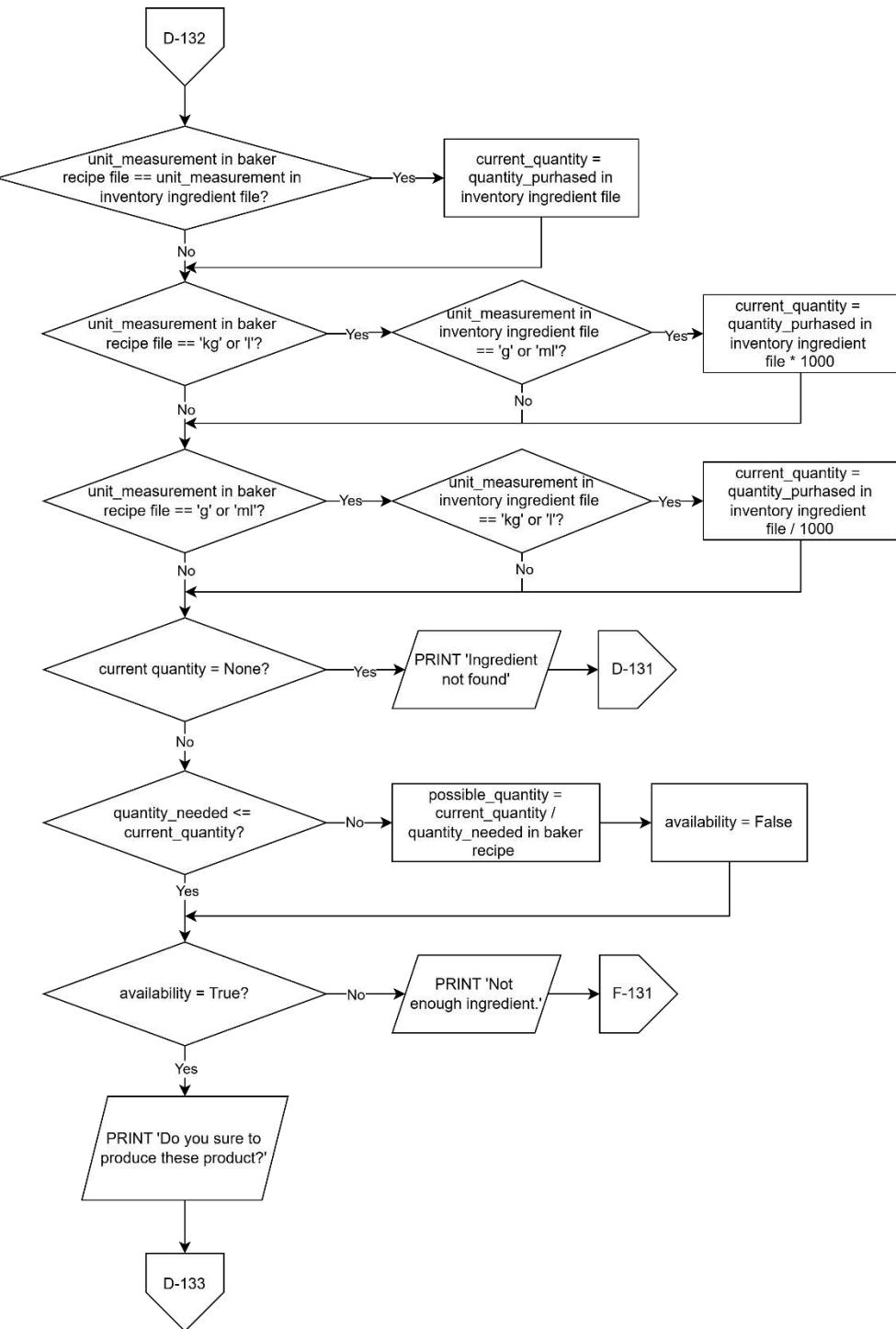


130

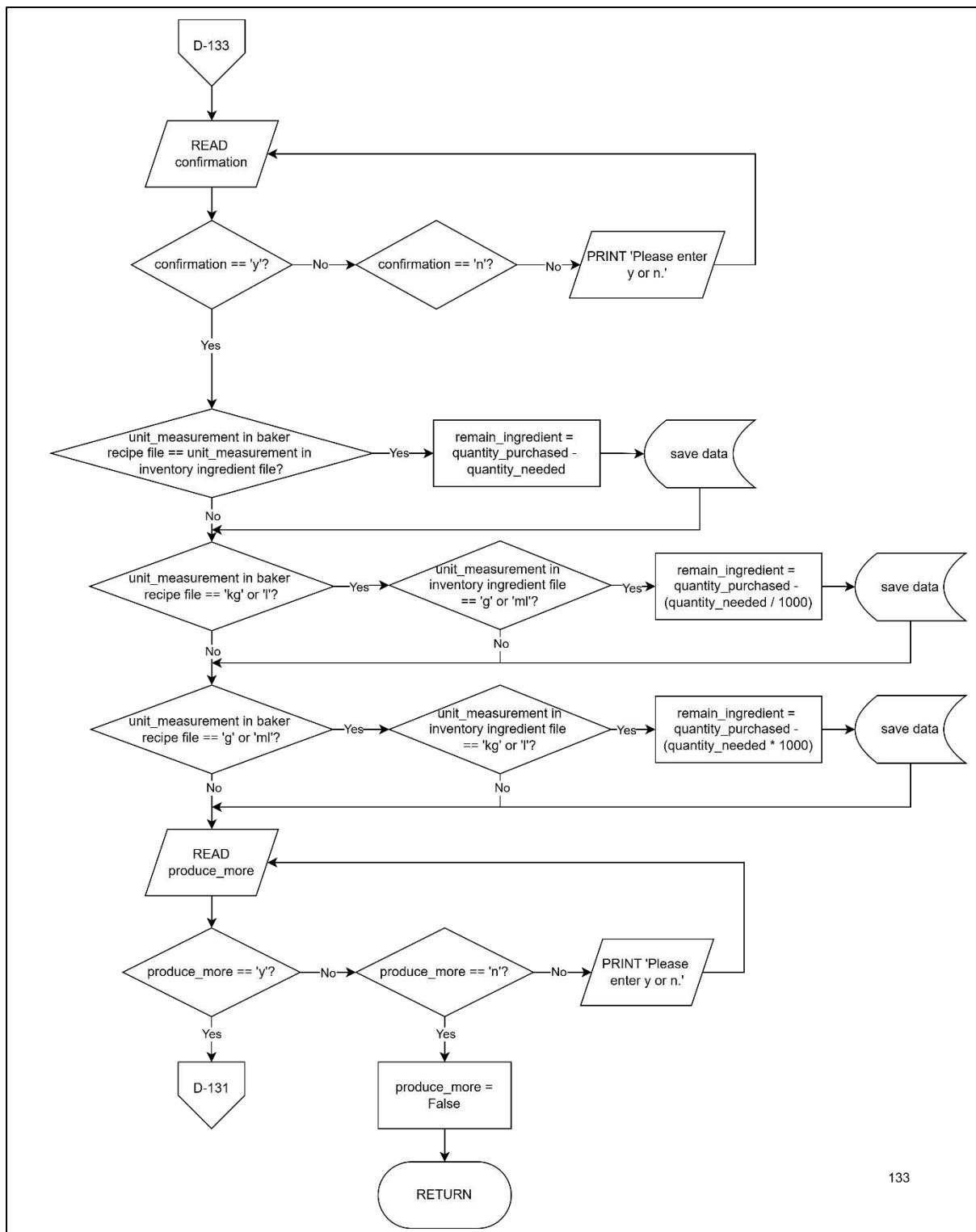
3.5.4 Inventory Check



131

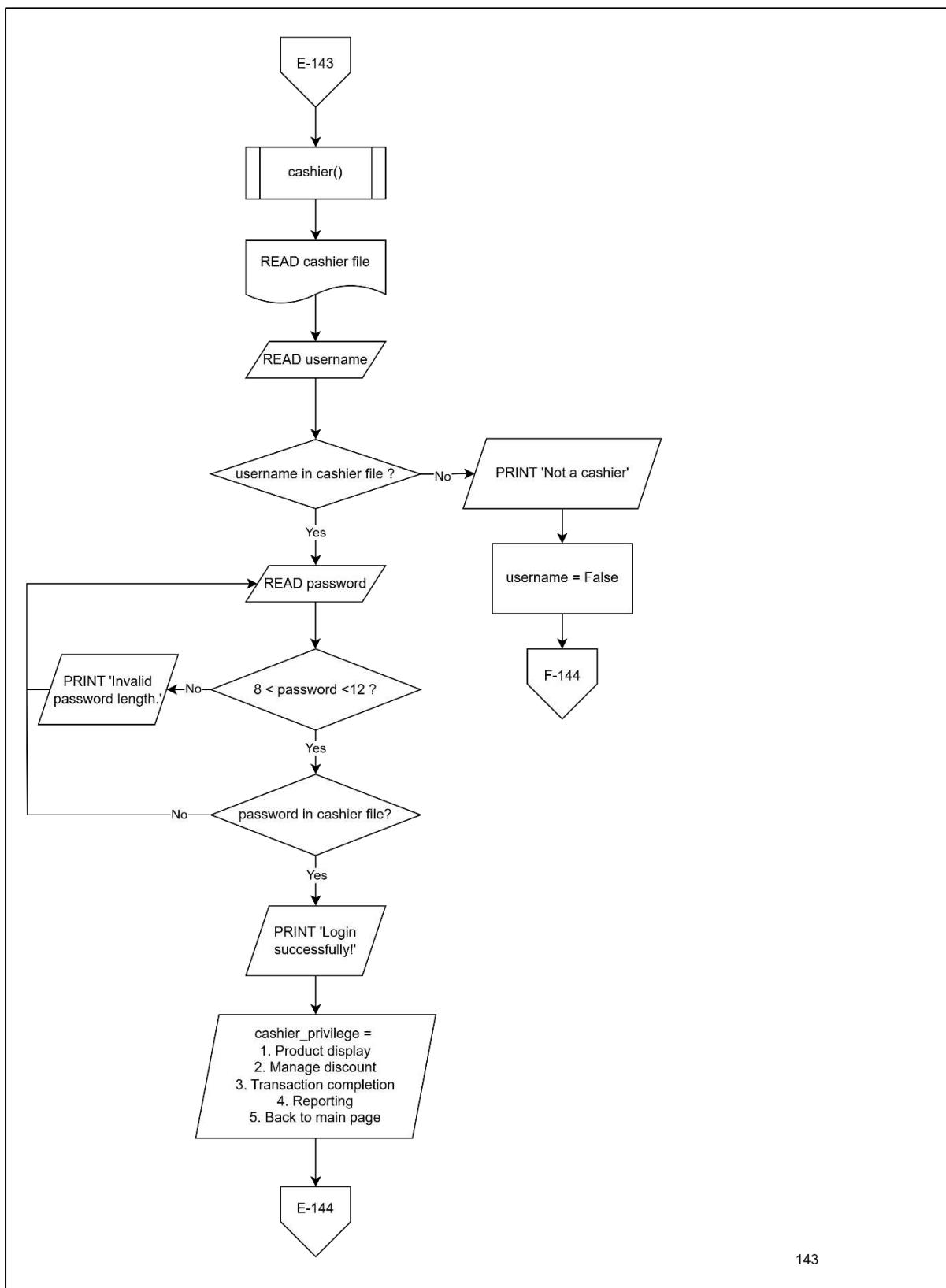


132

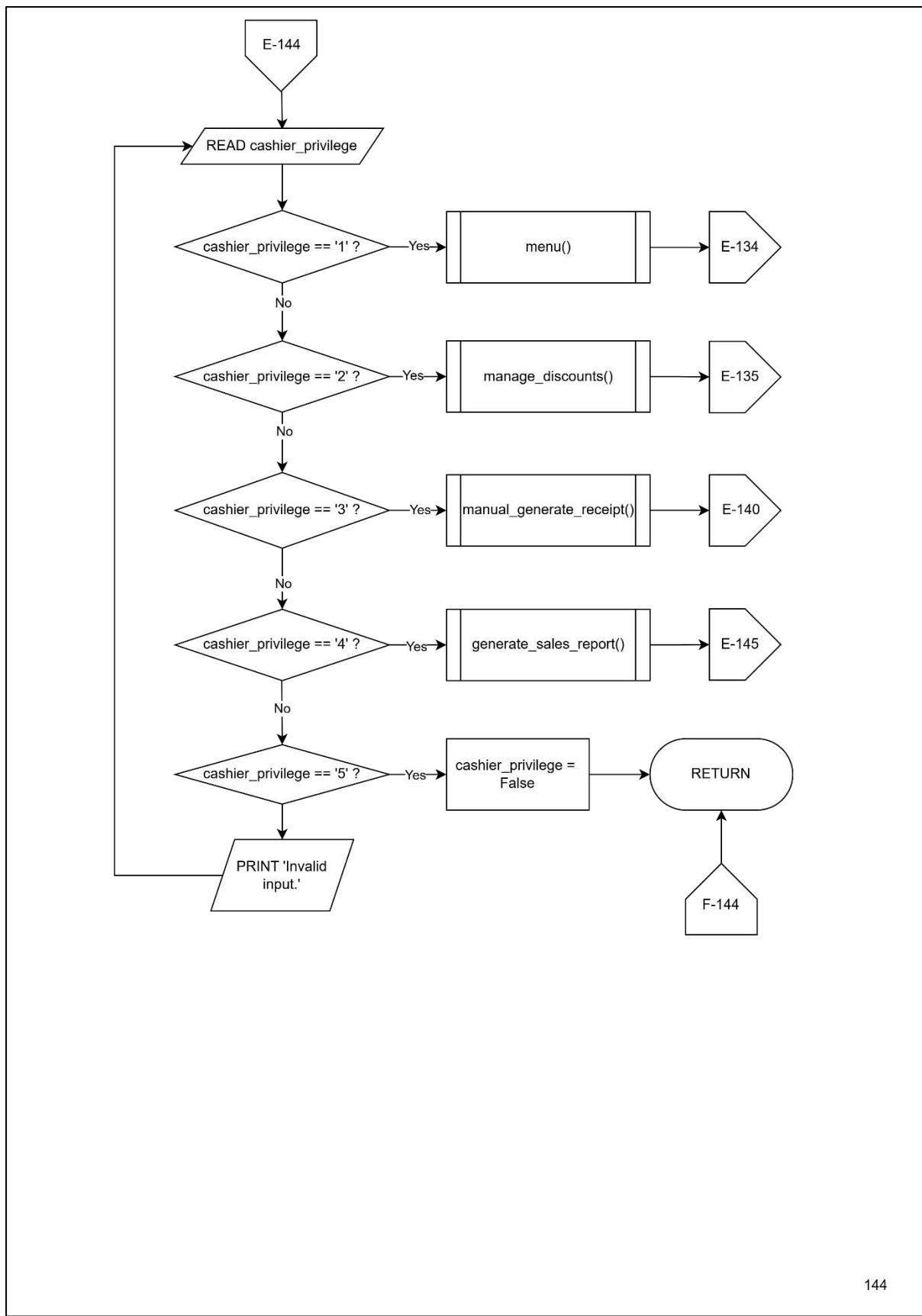


133

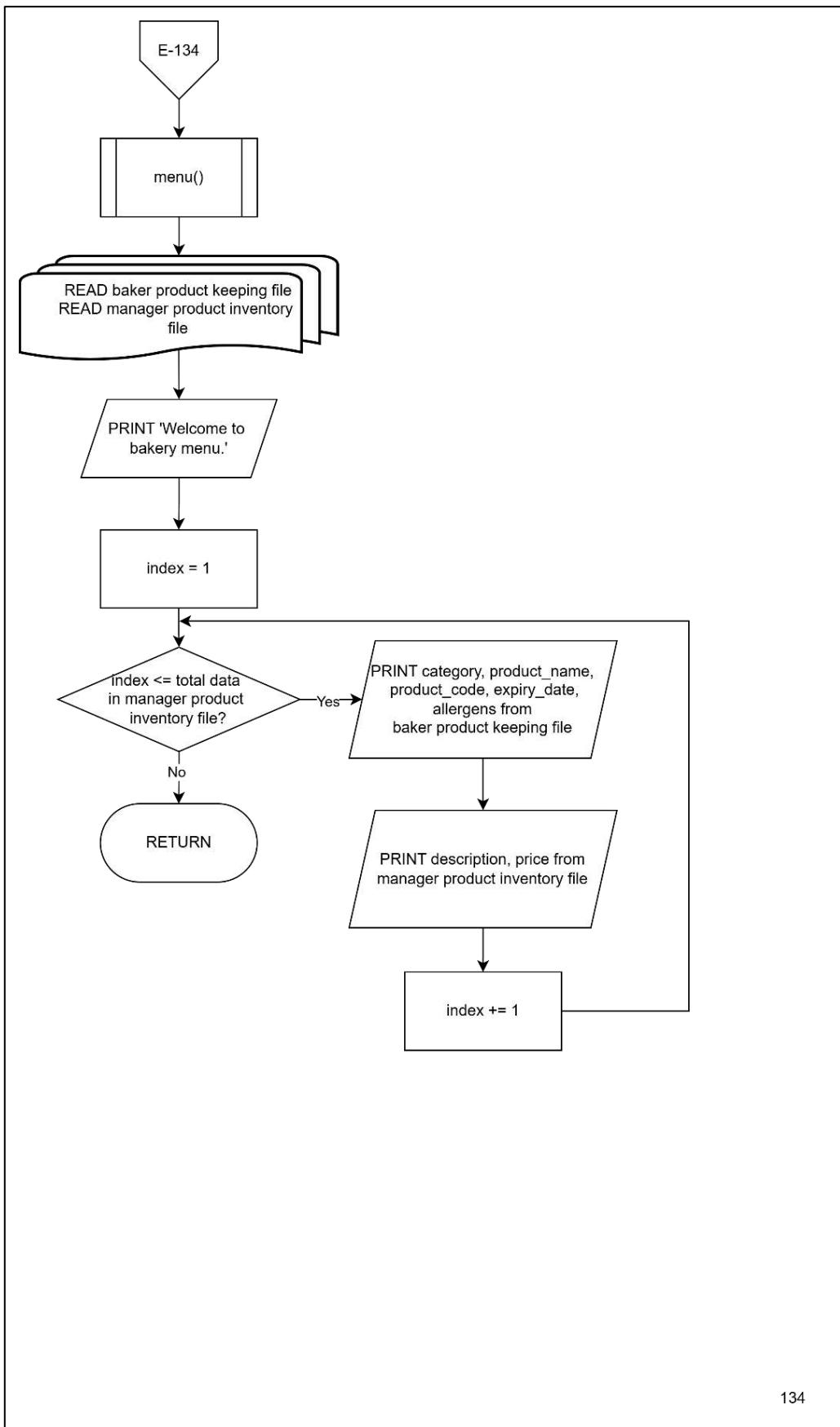
3.4 Cashier



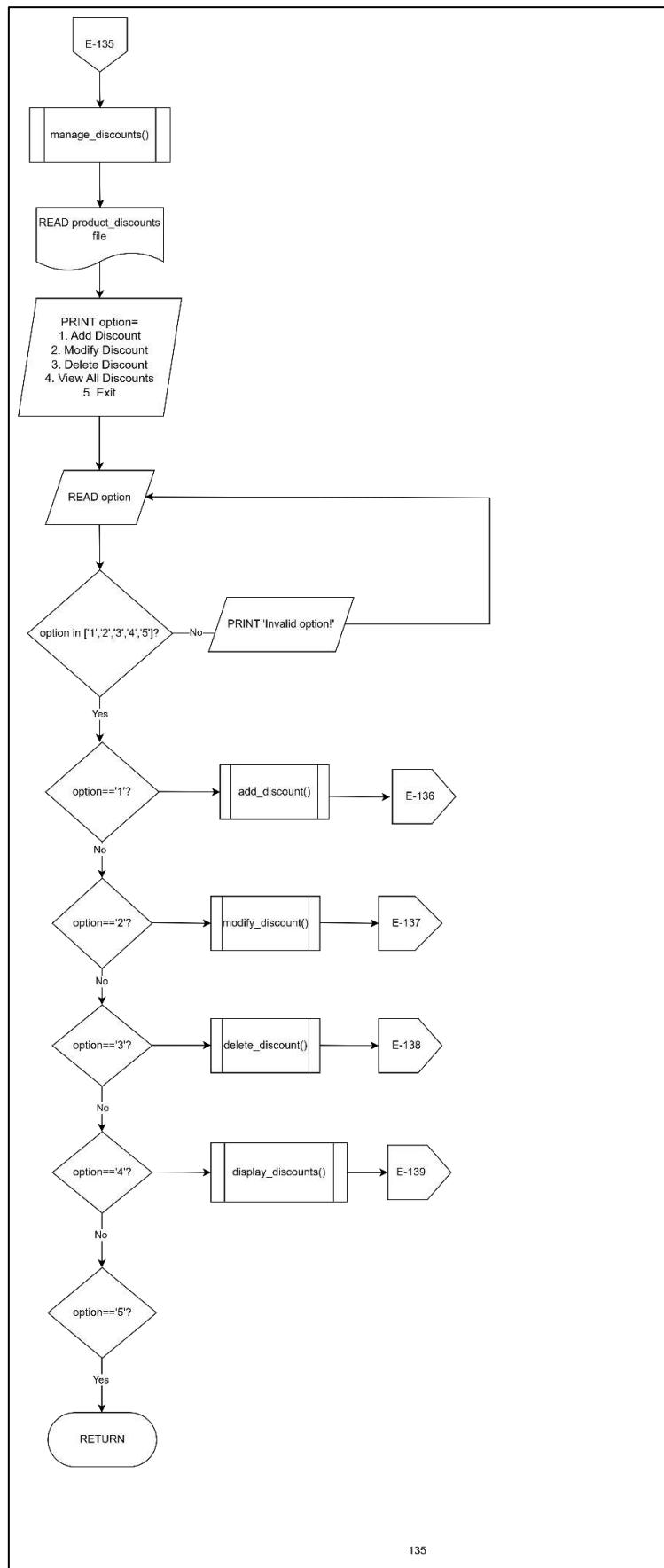
143

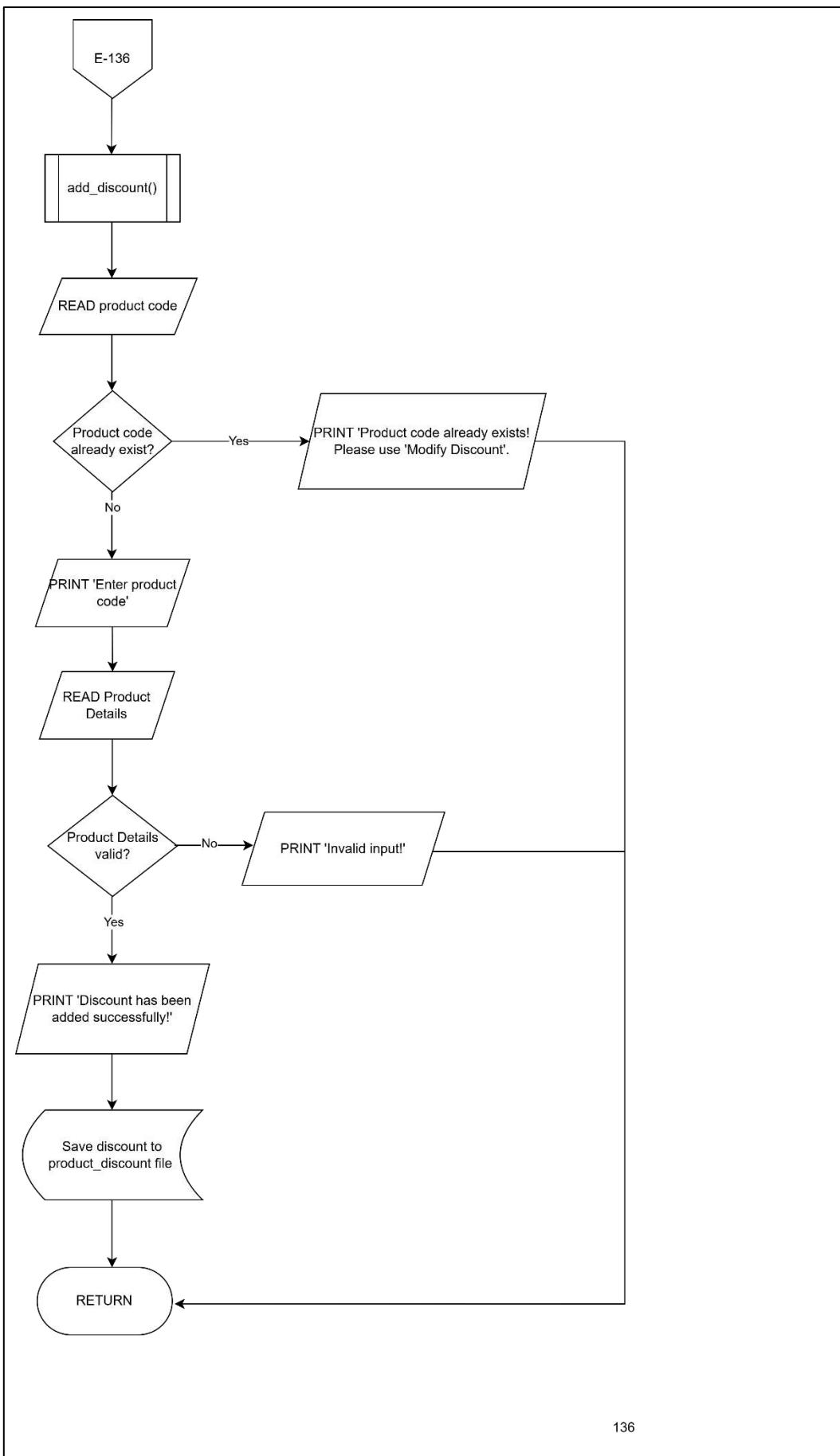


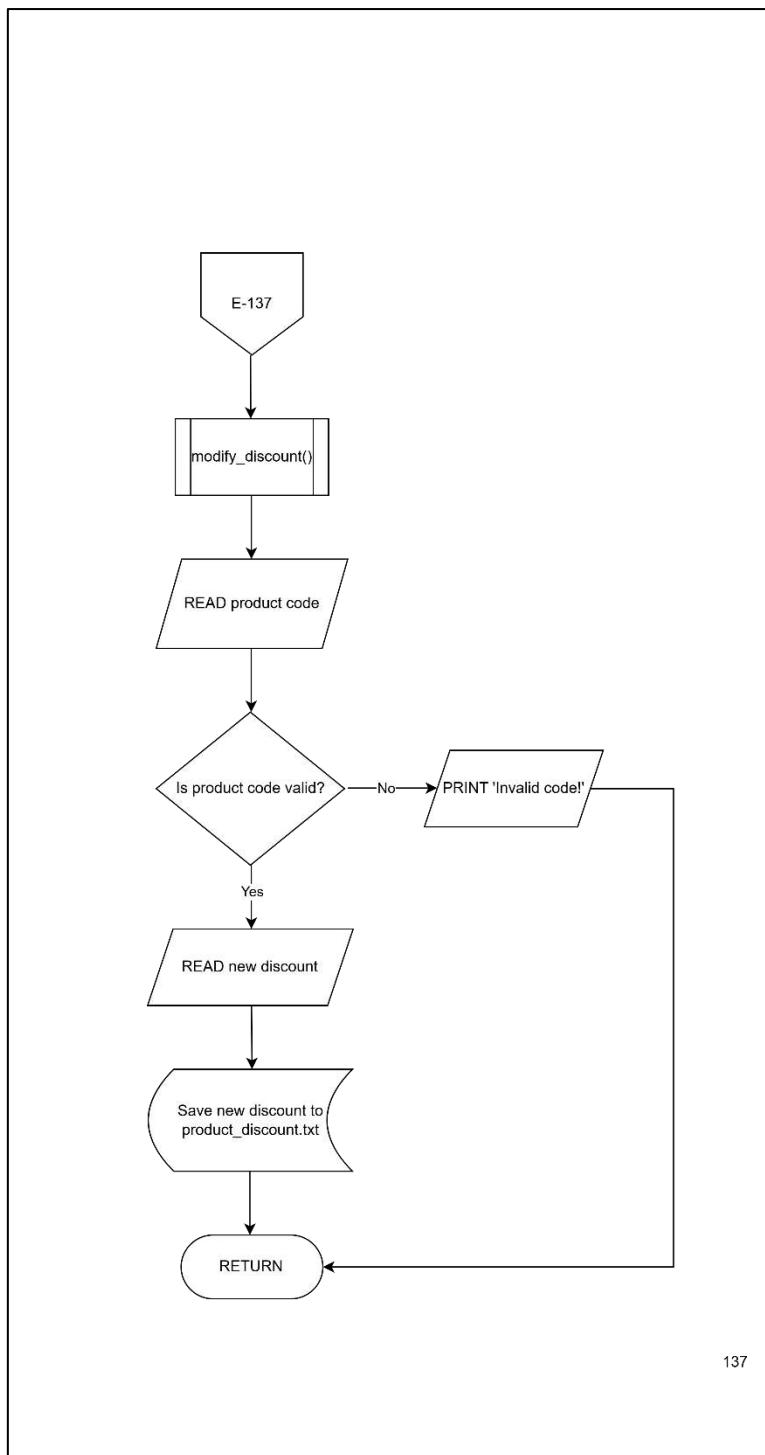
3.4.1 Product Display



3.4.2 Manage Discount

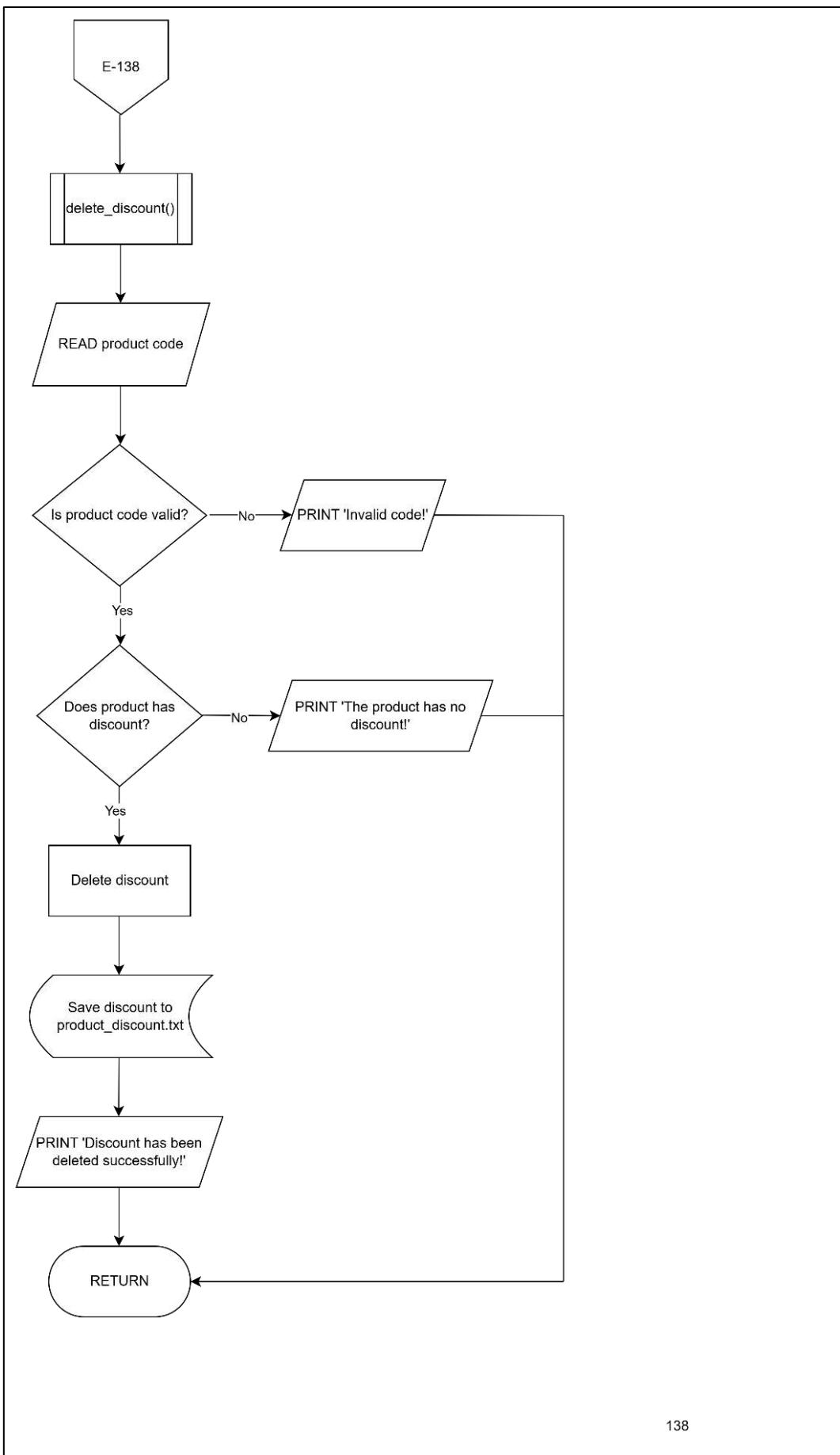


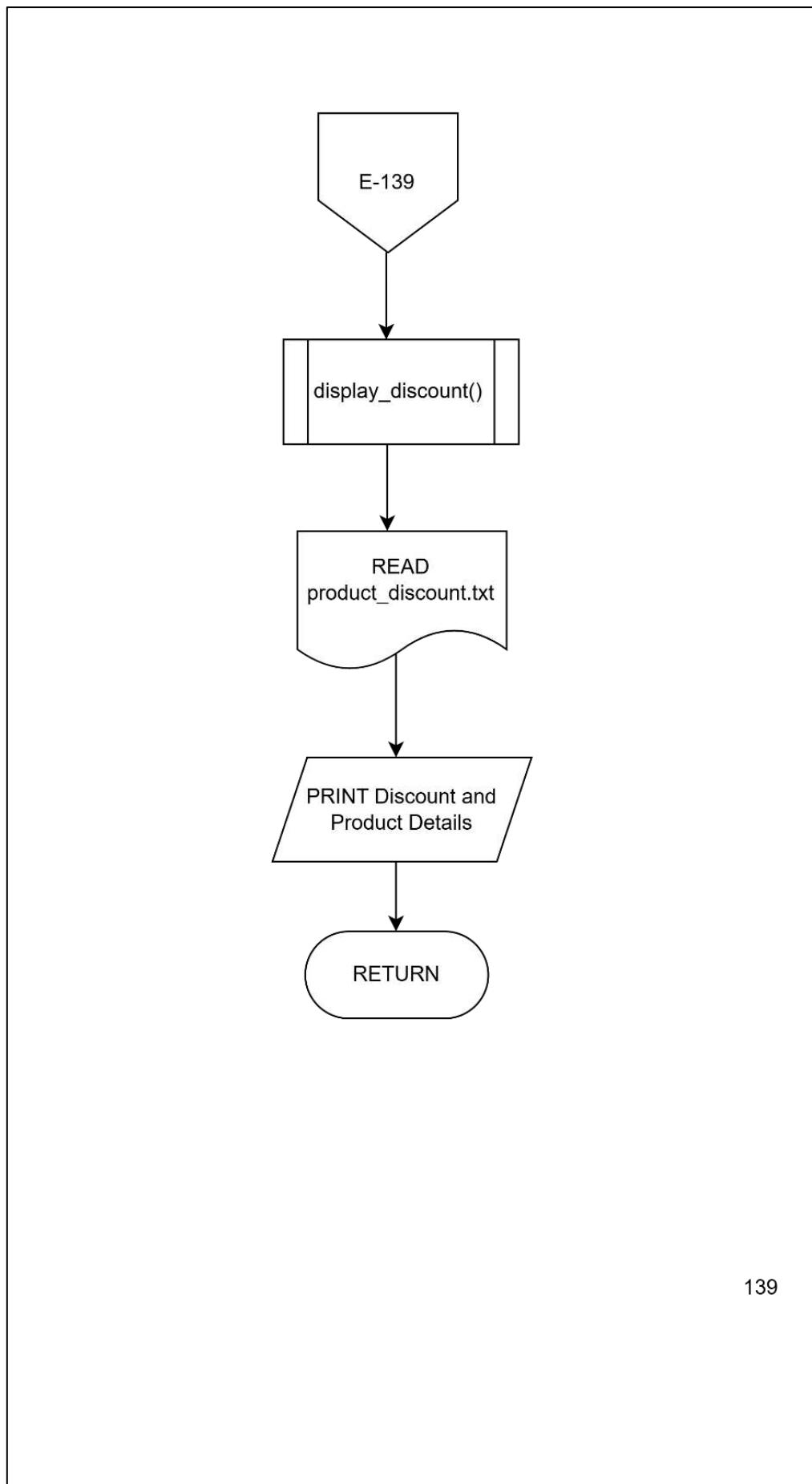




137

148

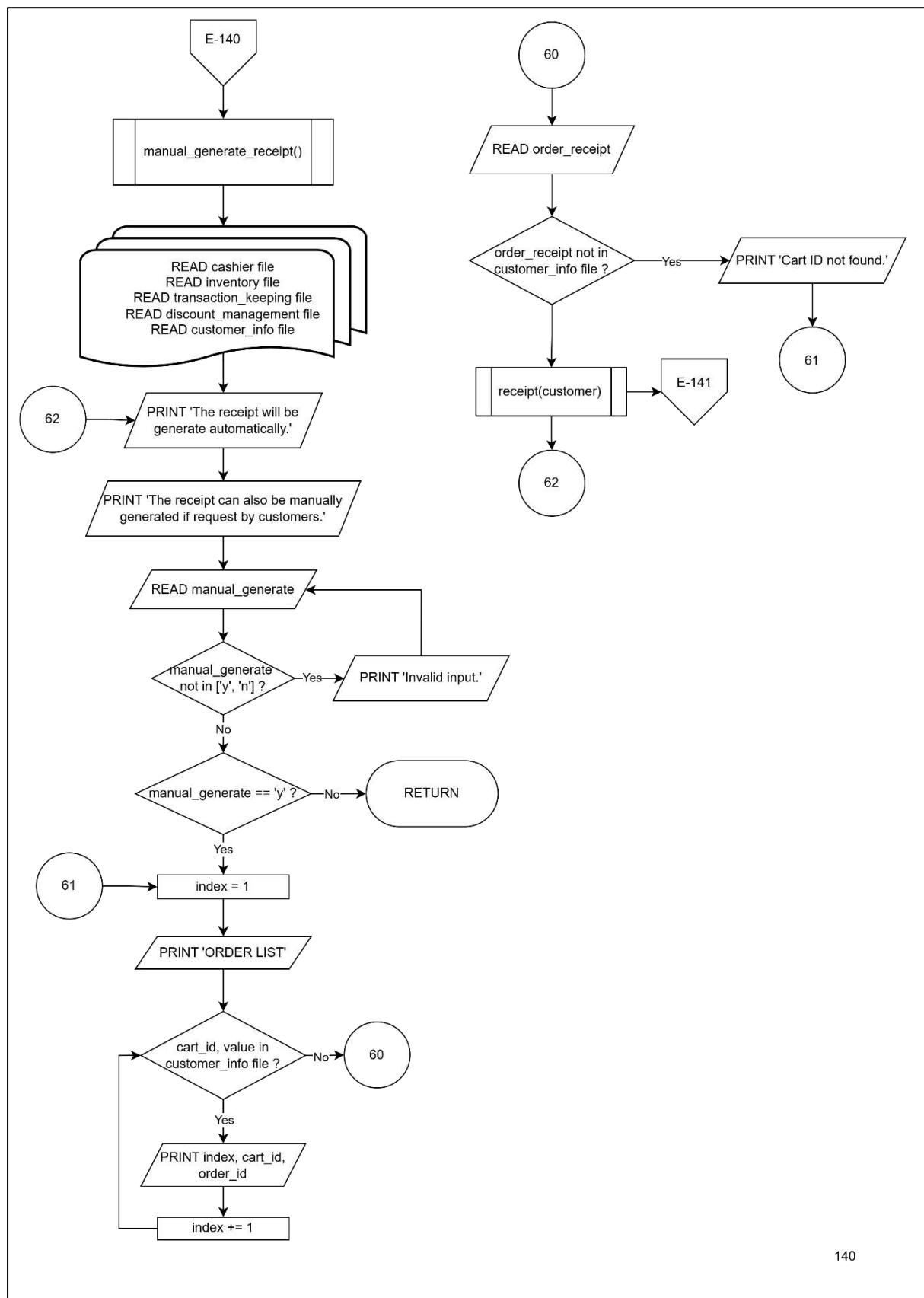




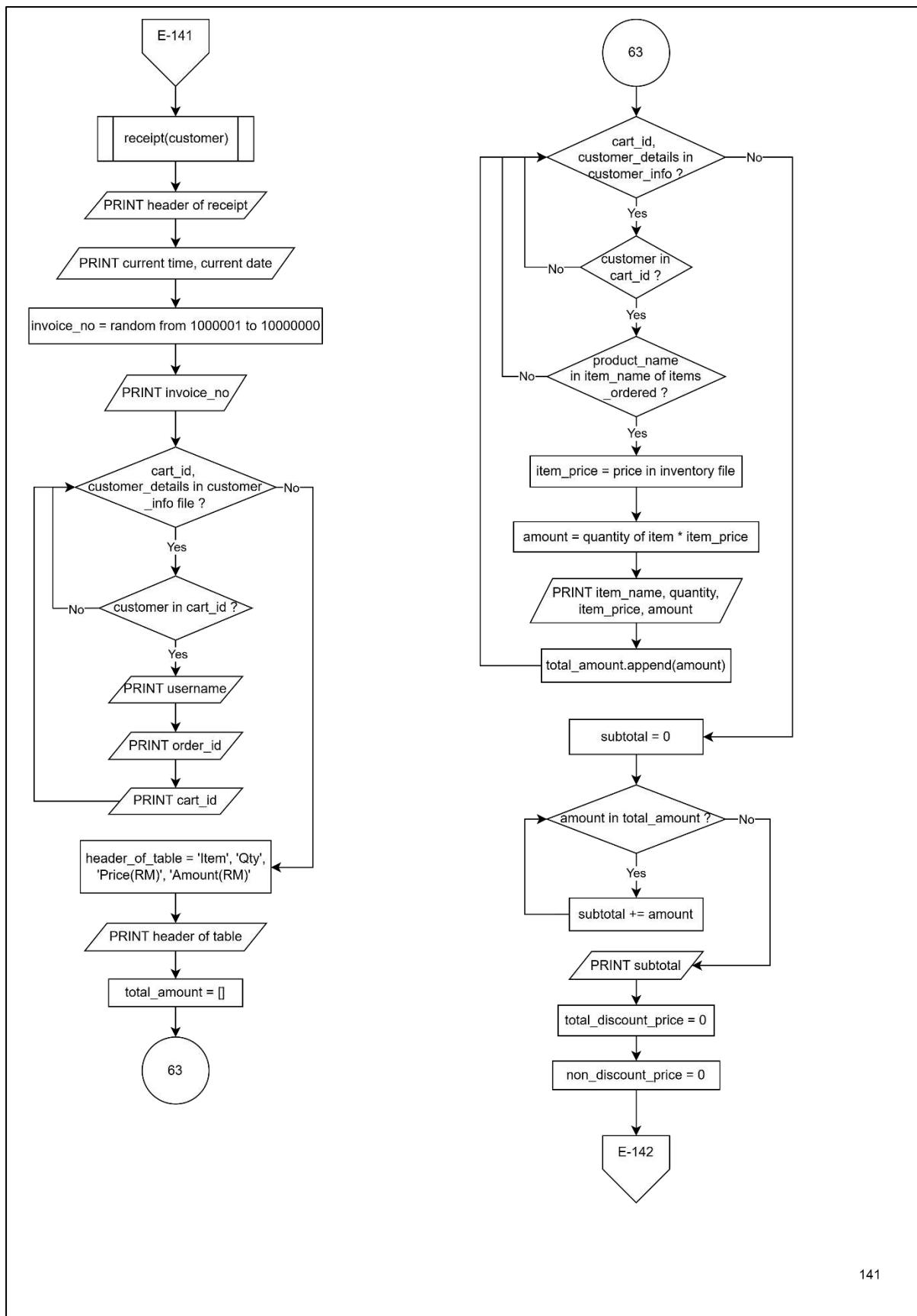
139

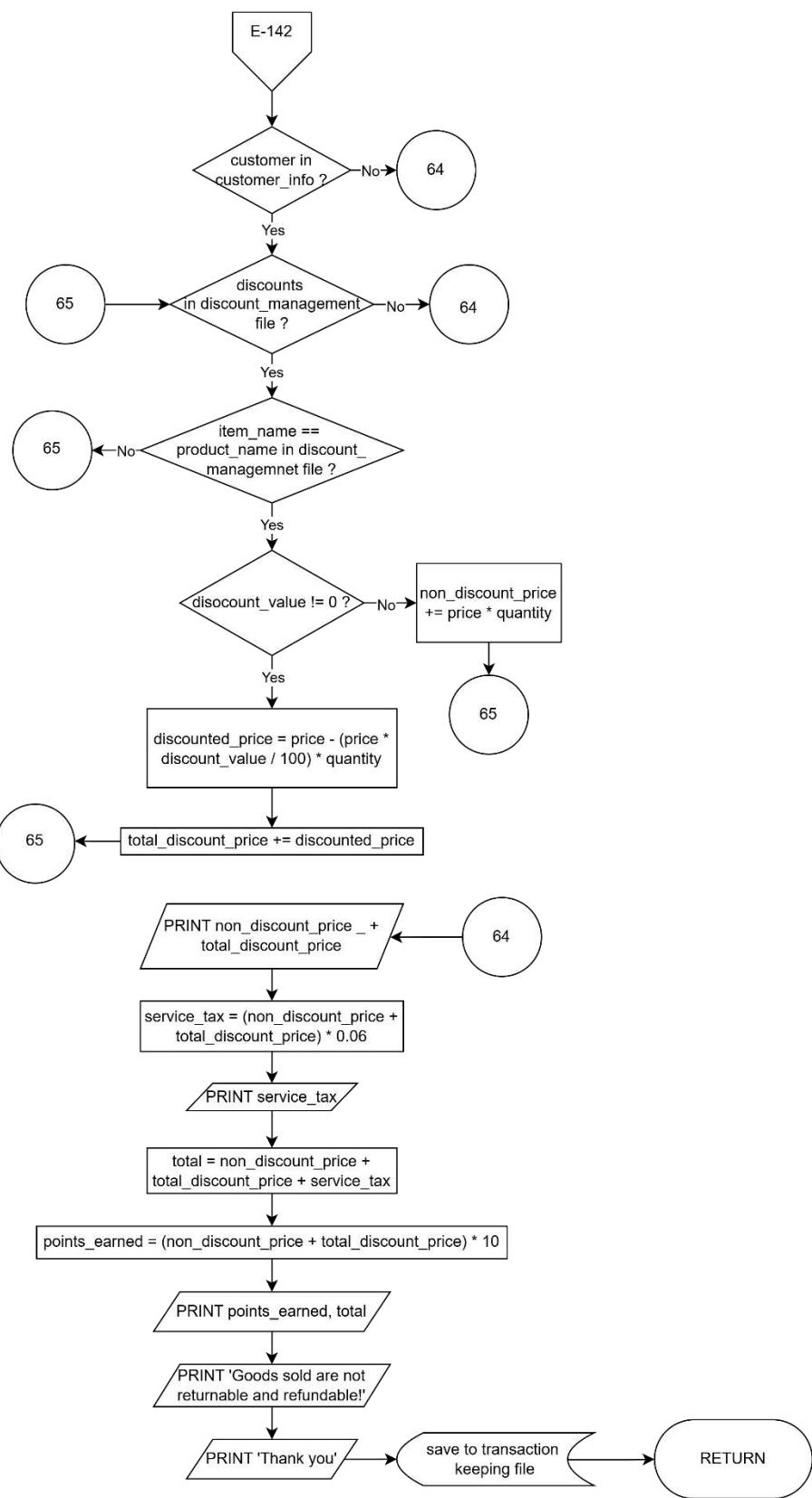
150

3.4.3 Transaction Completion



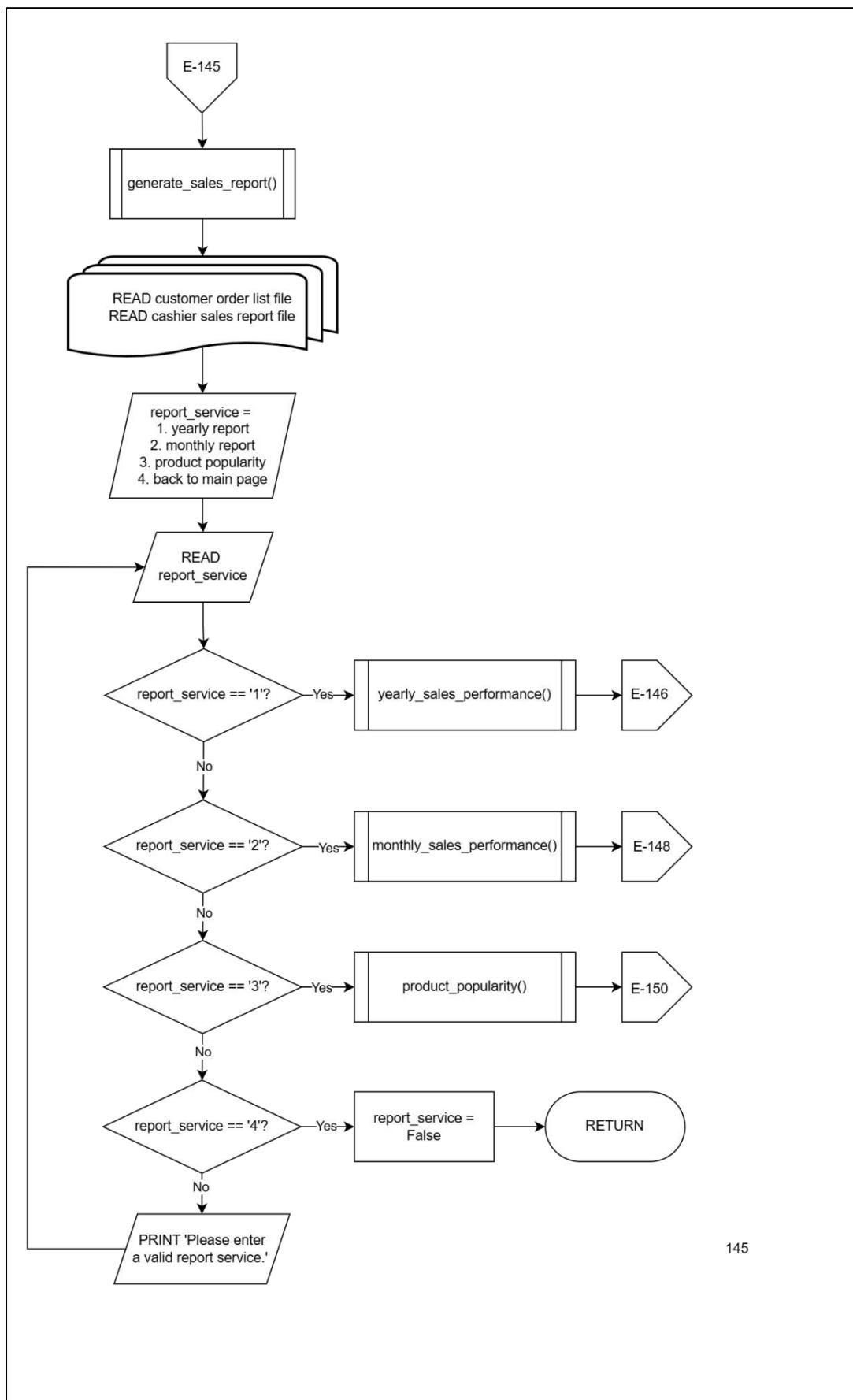
140



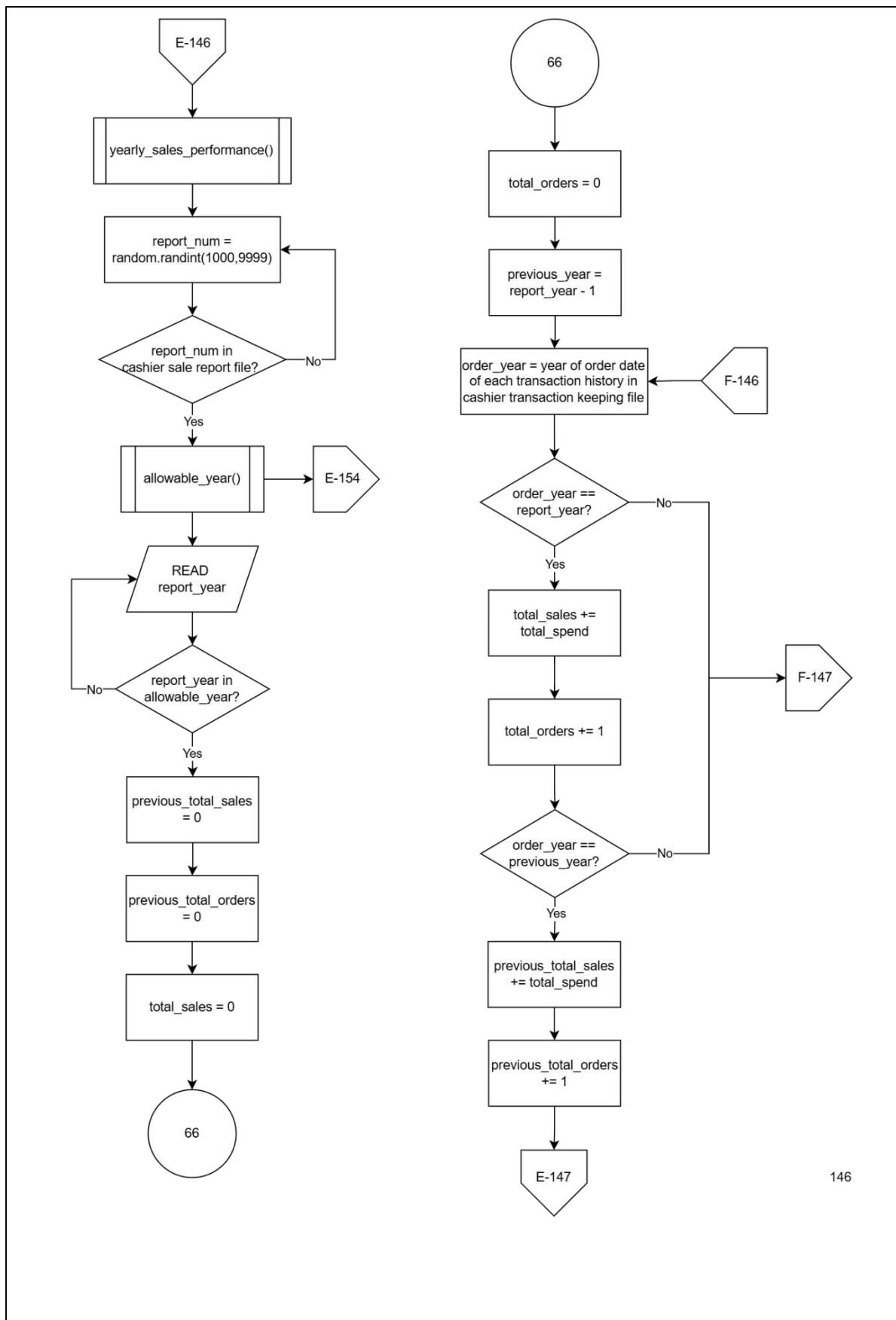


142

3.4.4 Reporting

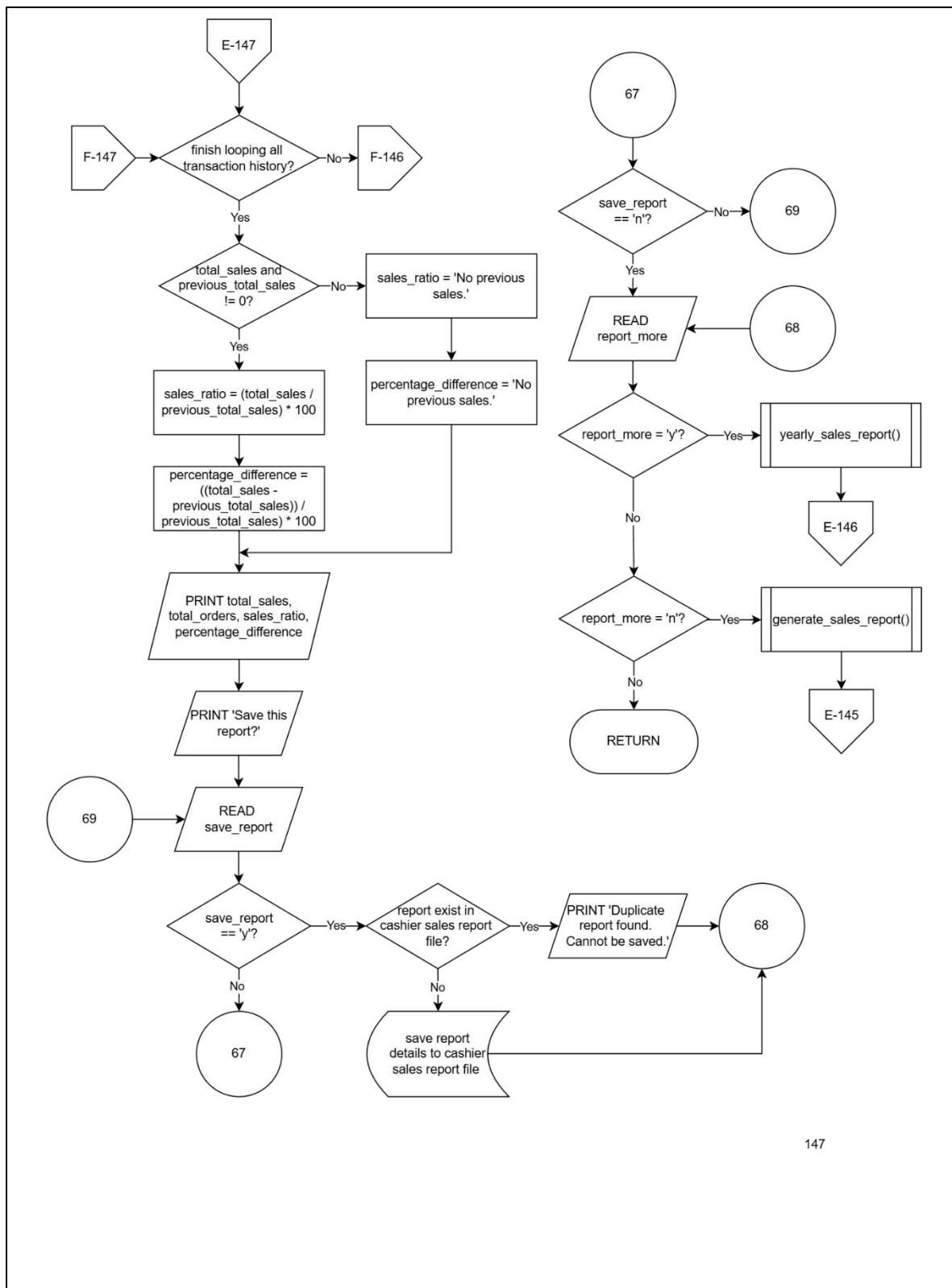


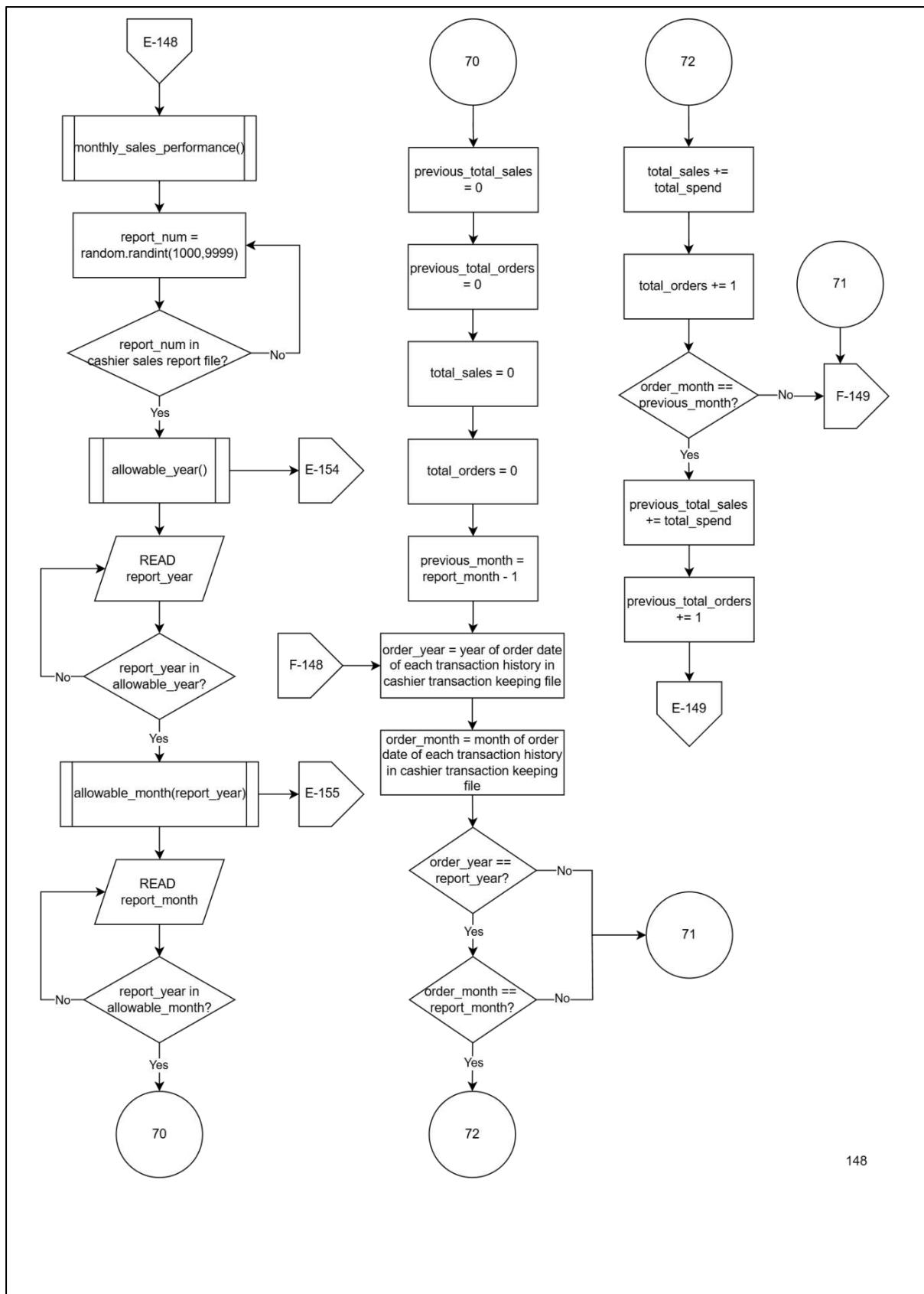
145



146

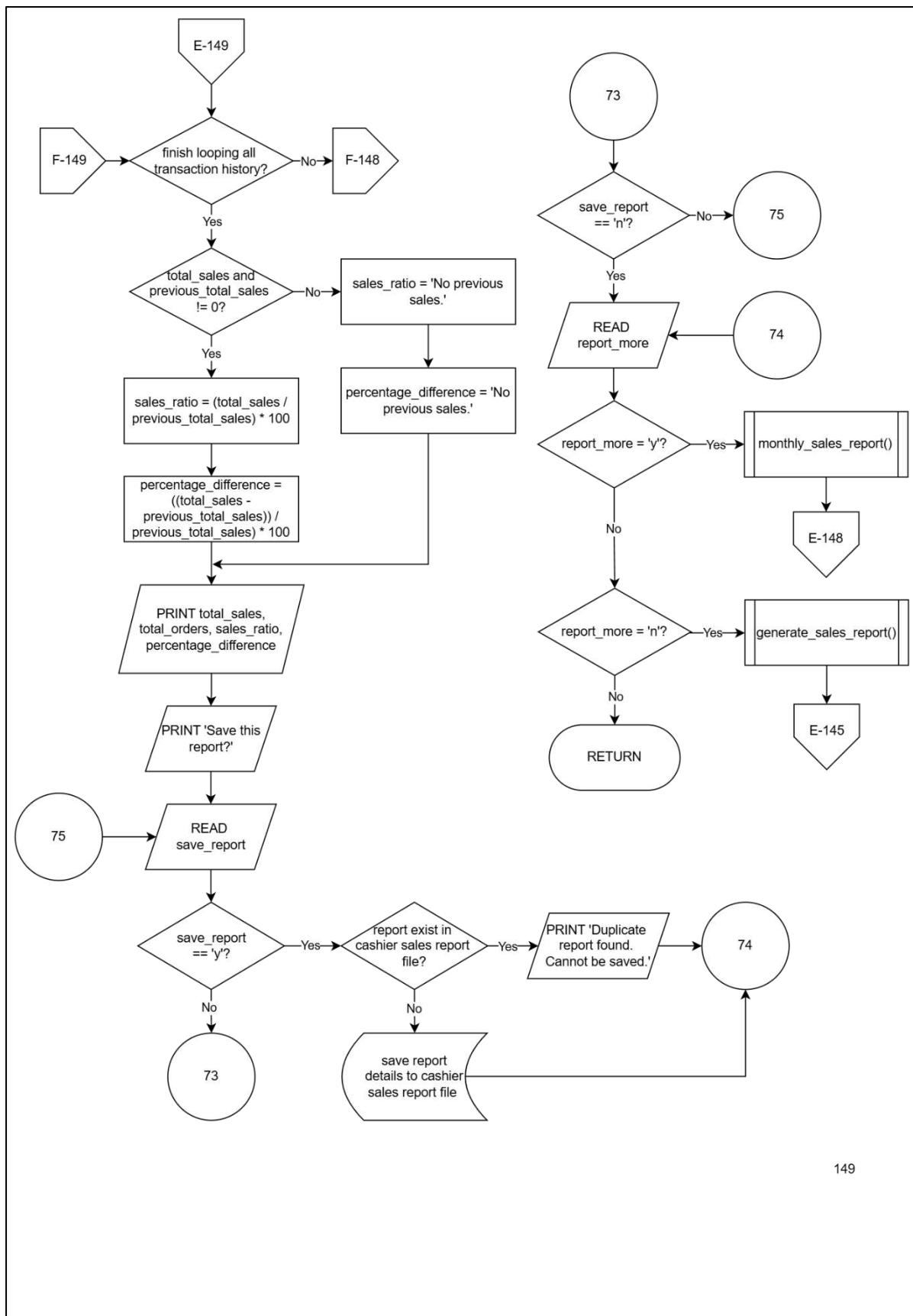
155





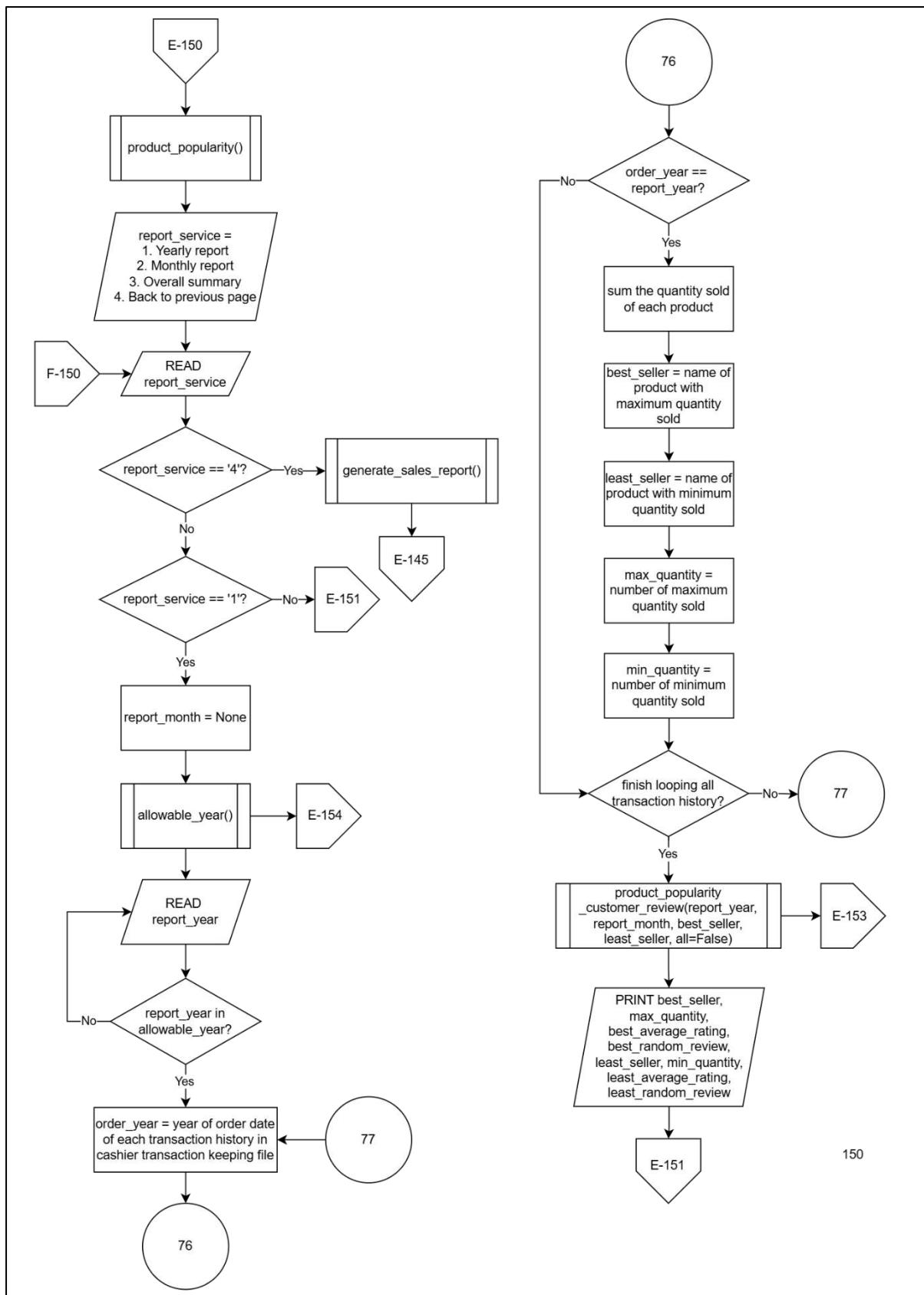
148

157

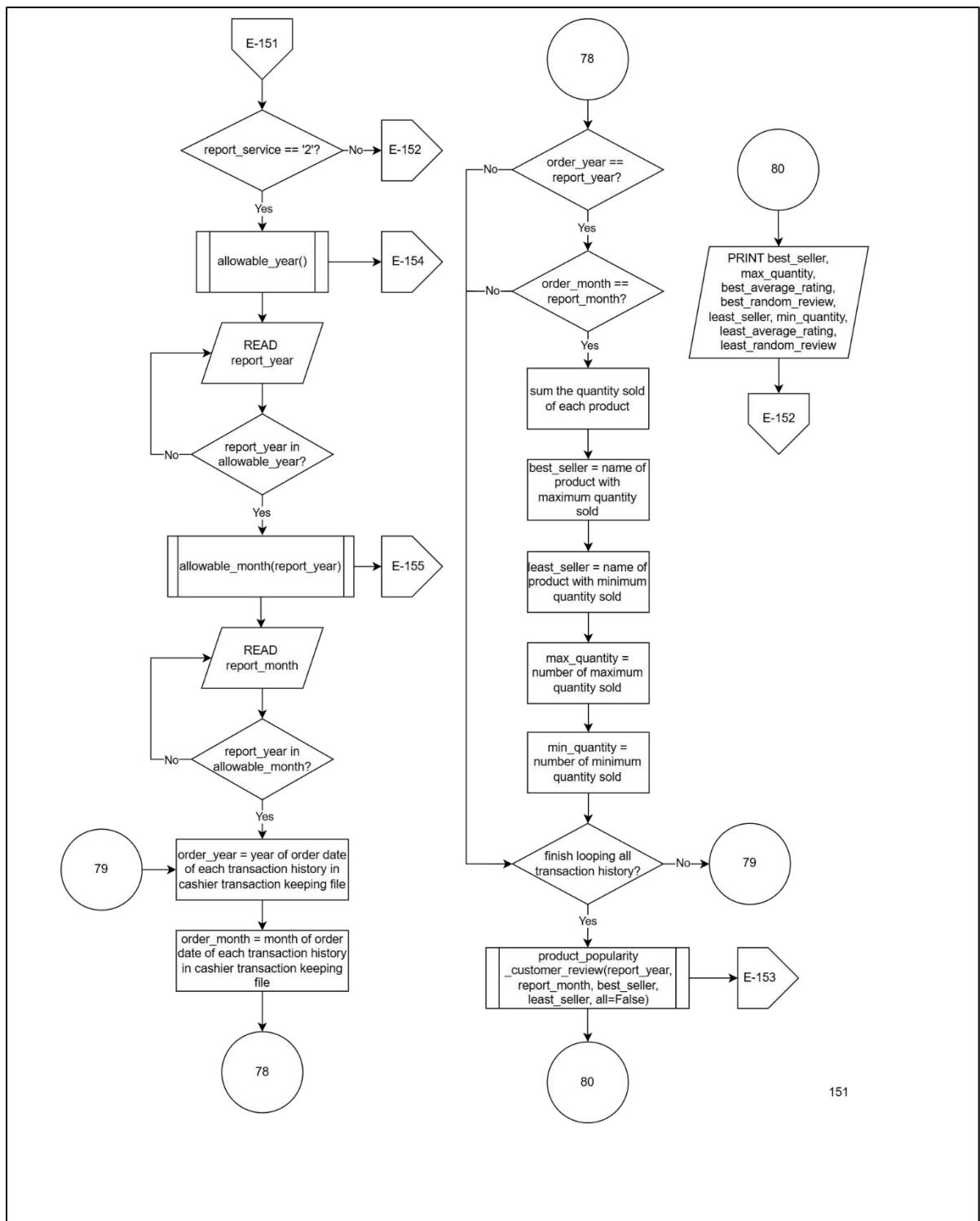


149

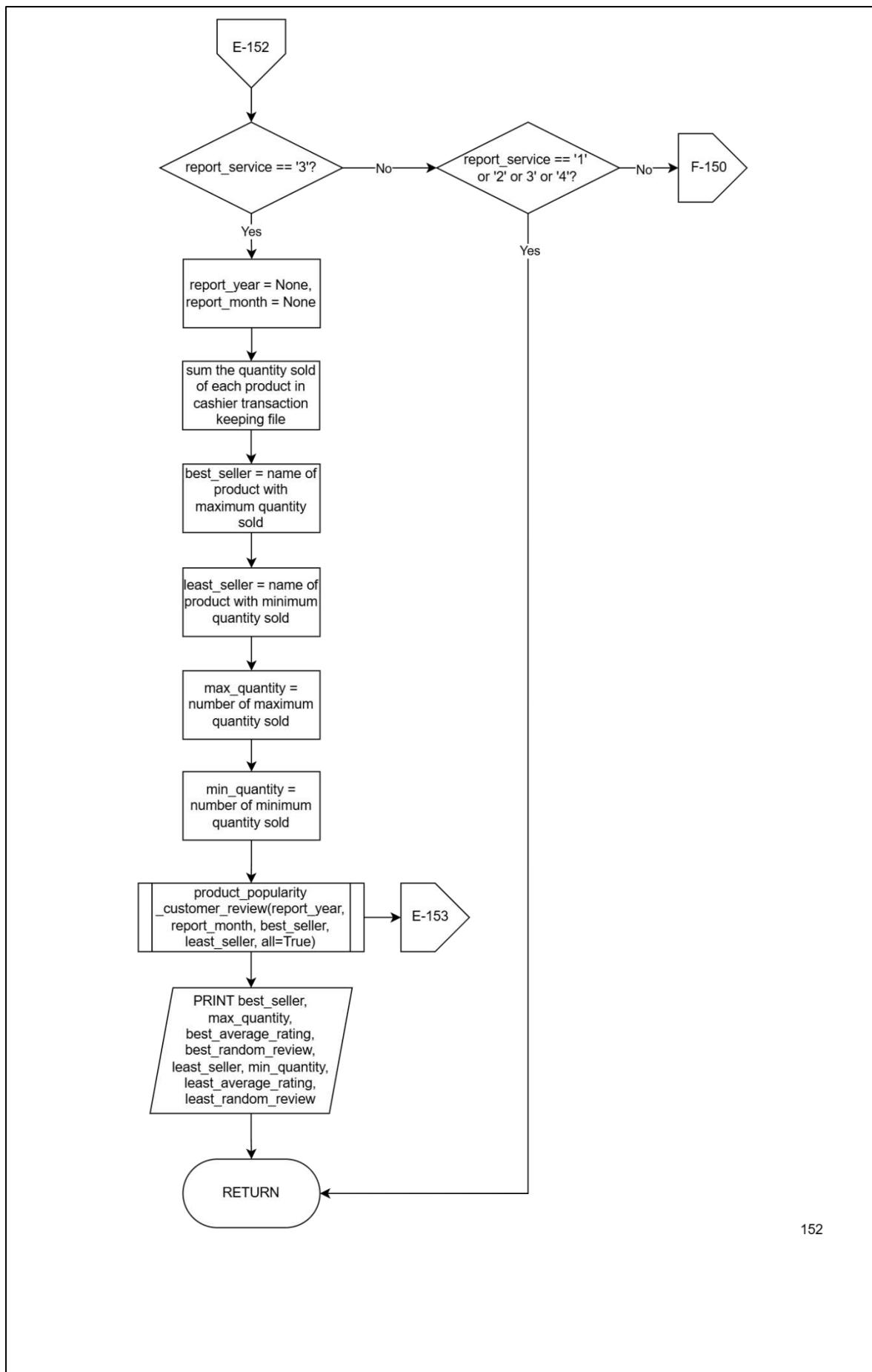
158



150

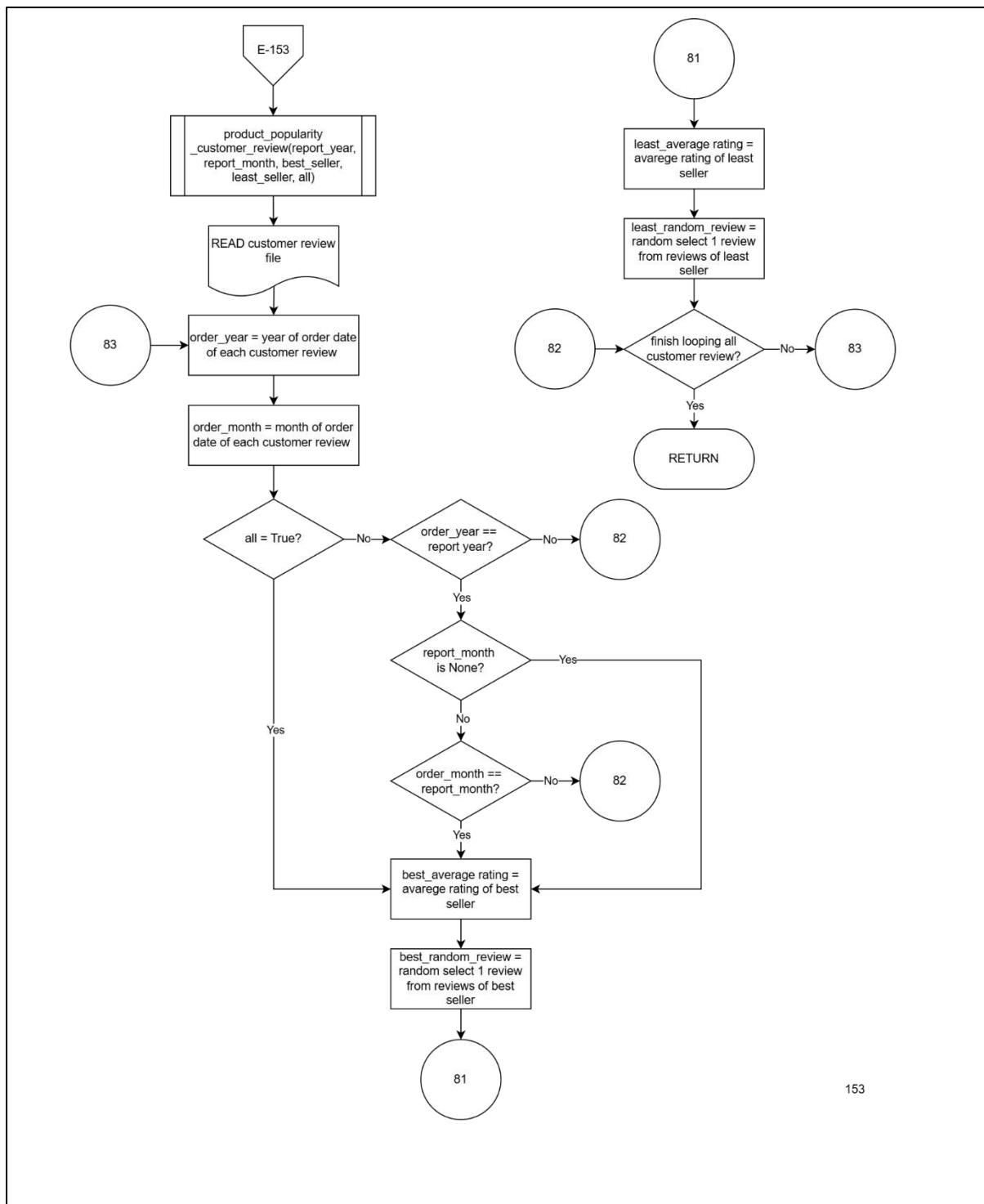


151

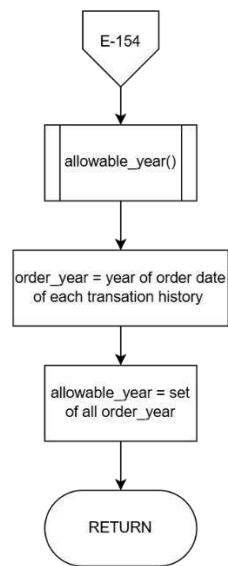


152

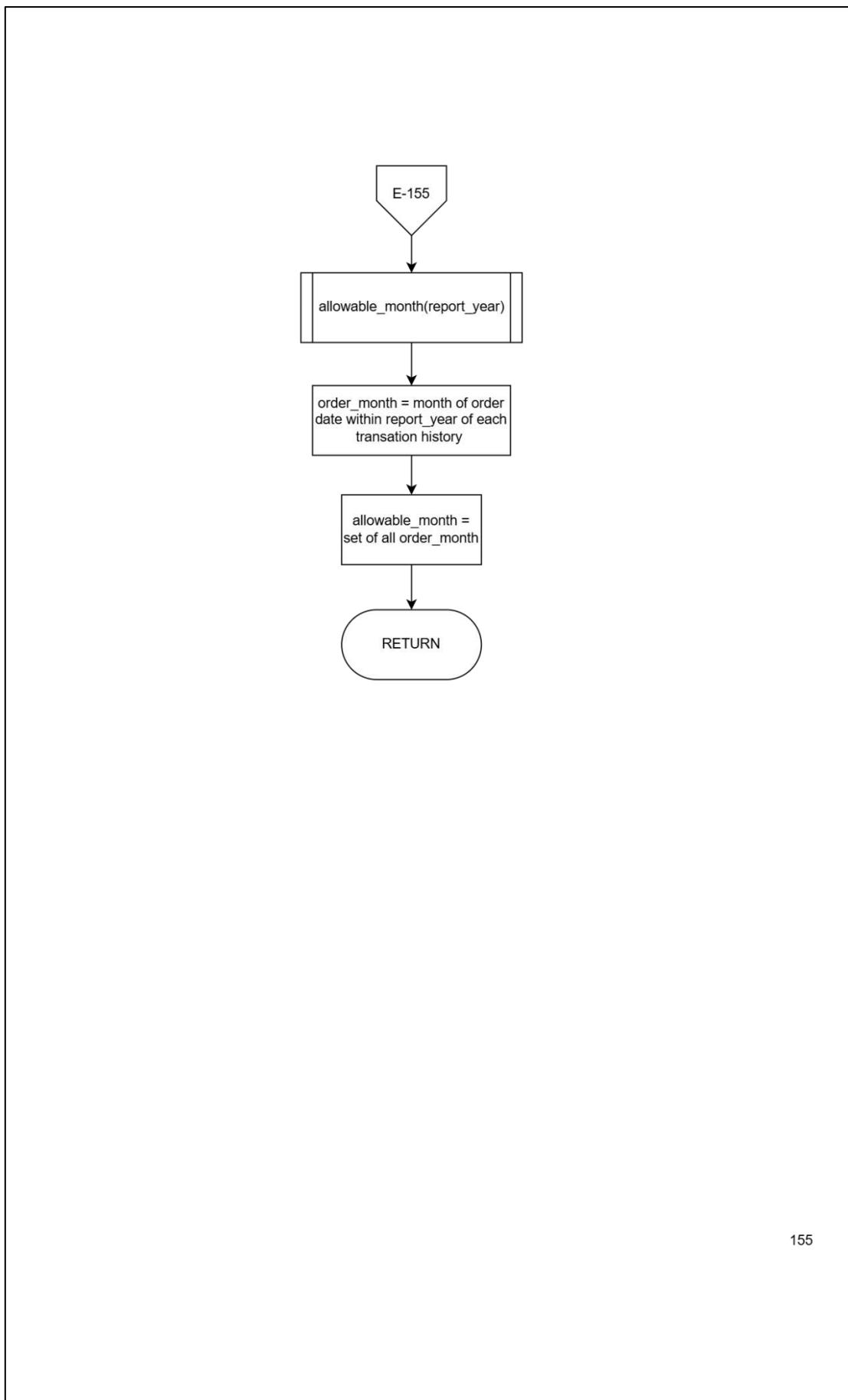
161



153



154



155

164

4.0 Concept of code snippets

The following code snippets with clear explanations below are the building blocks of Morning Glory Bakery Management System.

4.1 Auxiliary function

(a) File handling function

```
13  def load_data_from_manager():
14      try:
15          file = open('manager.txt', 'r') # open the file and read
16          content = file.read().strip() # strip() function is used to strip any unnecessary whitespaces
17          file.close() # close the file after reading
18          if content: # start to check if the file is not empty
19              try:
20                  return json.loads(content) # parse the content as json format into python dictionary and return the co
21              except json.JSONDecodeError:
22                  return {} # return empty dictionary if the content does not parse successfully
23          else:
24              return {} # return empty dictionary if the file is empty
25      except FileNotFoundError:
26          return {} # return empty dictionary if the file does not exist
27
28
29 # Define the function that saves information to the file
# Ng Yvonne
30 def save_info(manager_info):
31     file = open('manager.txt', 'w') # open the file to write
32     json.dump(manager_info, file, indent=4) # convert the dictionary into JSON format, 4 spaces indentation make it cle
33     file.close() # close the file after writing
```

Figure 4.1.1 File handling

These two file handling functions will be used in almost every module to enable programmer retrieve data from text files and save data to text files. All data save in the text files are dictionaries with Json format. In the first function, it opens the text file and set it to read mode. The outer exception handling is used to check if the file does not find, it will return as an empty dictionary instead of raising an error and cause the program to stop. The inner exception handling is used to check if the file is empty, it will also return an empty dictionary. In the second function, the file is open with write mode to save the data generated during program execution as Json format into text files. Json format can make the data look more organized and clearer. The indentation is specified for a clearer visualization. Both files will be closed properly after reading and writing.

Sample output:

```
1   {
2     "tsuki": {
3       "manager_username": "tsukiyaki",
4       "manager_password": "d0ugh8c5s",
5       "age": 45,
6       "gender": "female",
7       "contact_no": "123-3452621",
8       "email": "tsuki857@gmail.com"
9     }
10 }
```

Figure 4.1.2 Sample output of data store in text file

Figure 4.1.2 demonstrate a dictionary stored in a text file using Json format.

(b) Function that print content in centre

```
36 def printed_centered(info):
37     print('-' * 47)
38     side_space = (47 - len(info)) // 2 # determine how much blank space to leave
39     print(' ' * side_space + info + ' ' * (47 - len(info) - side_space))
40     print('-' * 47)
41
```

Figure 4.1.3 Function that print a content in centre

This function is used to showcase the content in centre with a specific width. The content is represented by the parameter ‘info’. In this case, the specific width is 47 characters. The function will calculate the blank space needed at both sides of the content to position it at the centre. Floor division is used to avoid float numbers, so that the result can multiply by space (‘ ’).

Sample output:

```
-----  
          MANAGER PRIVILEGE  
-----  
a. System Administration  
b. Order Management  
c. Financial Management  
d. Inventory Control  
e. Customer Feedback  
f. Notifications  
g. Back to Main page
```

Figure 4.1.4 Sample output of the content printed in centre with a specific width

In this case, the parameter of the function is ‘MANAGER PRIVILEGE’. It is displayed at the centre of the separator.

(c) Function that wrap data with a specific width

```
46     # Wrap the formatted data if it exceeds the space of 60 characters
47     2 usages  ± hxlum*
48
49     def wrap_data(formatted_product_data, width=65):
50         wrapped_lines = []
51
52         for data in formatted_product_data:
53             info_line = []
54             info = data.split()
55
56             for word in info:
57                 if len(' '.join(info_line + [word])) <= width:
58                     info_line.append(word)
59                 else:
60                     wrapped_lines.append(' '.join(info_line))
61                     info_line = [word]
62             if info_line:
63                 wrapped_lines.append(' '.join(info_line))
64
65
66     return wrapped_lines
```

Figure 4.1.5 Function that wrap data with a specific width

The function initializes an empty list named wrapped_lines to store the final wrapped lines. Next, it gets a list of strings from formatted_product_data and split each string to individual words. It then iterates through each word to form lines, by checking if adding the current word will exceed the limited width (65 characters) through summing up the character count of current word, spacing for each word and the total character of existing words in info_line. If adding the words will not exceed 65 characters, it will append the words to a list, info_line that initialize to empty in the beginning. If adding the word will exceed the width limit, it will join the words in info_line using spacing and append it to wrapped_lines. Then, it starts a new info_line with current word. After processing each word, any remaining words in info_line is joined and added to wrapped_lines. Last, the function will return the wrapped_lines list.

Sample Output:

```
CK858 - Tiramisu
Price: RM 8.50
Best Before: 25-09-2024
Allergen: Coffee, Biscuit
    • Coffee-soaked ladyfingers with creamy mascarpone and blueberry
      toppings

MAT030 - Matcha Layer Cake
Price: RM 12.00
Best Before: 08-11-2024
Allergen: No
    • A delicate, multi-layered cake with earthy matcha flavor and a
      creamy, smooth finish.
```

Figure 4.1.6 Sample output of wrapped text

The output shows that the function will accurately wrap the text into lines that do not exceed 65 characters, if exceed, it will open a new line. This approach significantly improves the readability and consistency of printed text.

(d) Function that print two sets of data in pairs

```
68     # Format to print data retrieve from category_group dictionary side by side
69     #usage : hxium+1
70     def print_in_column(info1, info2, width=65):
71         max_line = len(info1)
72         if len(info2) > max_line:
73             max_line = len(info2)
74         new_info1 = list(info1) + [""] * (max_line - len(info1))
75         new_info2 = list(info2) + [""] * (max_line - len(info2))
76
77         for line1, line2 in zip(new_info1, new_info2):
78             print(f'{line1.ljust(width + 8)}{line2.ljust(width)}')
```

Figure 4.1.7 Function that print two sets of data in pairs

The function initializes the width to 65 characters. The function will then calculate the total number of lines in the list info1 and info2, to determine the maximum lines (max_line). Then,

it will extend the shorter line with empty string until it reaches `max_line` to ensure each list have equal number of lines. It then iterates through both lists simultaneously, left justified the entries of first list with padding of `width + 8` characters, and left justified the entries of second list with padding of `width`. Next, it prints each corresponding pairs of formatted entries on a single line.

Sample Output:

PA932 - Apple Pie Price: RM 5.00 Best Before: 06-09-2024 Allergen: Apple 👉 A mouthwatering dessert with tender filling covered by crispy crust	PA325 - Bello Minion Biscuit Price: RM 3.00 Best Before: 09-04-2024 Allergen: Coffee, Mango 👉 Cute and attractive appearance with mango filling
------------------------------------------------------------------------------------------------------------------------------------------------------------	-------------------------------------------------------------------------------------------------------------------------------------------------------------

Figure 4.1.8 Sample output that print two sets of data in pairs

The output shows that a clear two-column display is performed, where each row contains one items from each list. Extra spaces are also added if the number of lines of both lists are not equal, in order to align the columns consistently and uniformly.

(e) Function that validate empty entries, alphabetical and date

```
72 # validate and print error message when meet empty entries
23 usages  ▲ Ng Yvonne
73 def validation_empty_entries(info):
74     if info:
75         return True
76     else:
77         print('\n ! Please enter something...\n')
78         return False
79
80
81 # validate input that store a list is alphabet only
1 usage  ▲ Ng Yvonne +1
82 def validation_list_alphabet_only(info):
83     for item in info:
84         if item.replace(" ", "").isalpha():
85             return True
86         else:
87             return False
88
89
90 # validate whether user input is in correct date format
3 usages  ▲ Ng Yvonne
91 def validation_date(info, date_format='%d-%m-%Y'):
92     try:
93         datetime.strptime(info, date_format)
94         return True
95     except ValueError:
96         return False
```

Figure 4.1.9 Function that validates user input

To prevent duplication, the following validation functions are created for reusability across different inputs. The validation_empty_entries() function is used to check whether user input have a value. If yes, it will return True, allowing the loop to break. If no, it will print error message and return False to continue the while loop until user enter a value.

The validation_list_alphabet_only(info) function is used to check whether the value in a list contain only alphabetic characters. It eliminates any spaces in the value, and ensure it only contain alphabet. If yes, it will return True, allowing the loop to break, if no, it will return False to continue the while loop until user enter a valid input.

The validation_date(info, date_format = ‘%d-%m-%Y’) function is used to check whether user input is in date format. It converts the input which initially a string to a date object, and check

if it is valid. If yes, it will return True, allowing the loop to break, if no, it will return False to continue the while loop until user enter a valid date.

Sample Output:

```
2. Product Name      :  
  
! Please enter something...  
  
2. Product Name      : Minion biscuit  
3. Product Code      :
```

Figure 4.1.10 Sample output of function that validate empty entries

```
4. Date of Production : 11-11-1111  
  
+-----+  
| ▲ Please enter a valid date of production based on the date given above. |  
+-----+  
  
4. Date of Production : 09-04-2024  
5. Shelf Life (_ days) : |
```

Figure 4.1.11 Sample output of function that validate date

```
9. Allergens          : rhg5 6jd jj4  
  
+-----+  
| ▲ Please enter a valid allergen. (Cannot contain any digits and special characters.) |  
+-----+  
  
+-----+  
| ⚡ If there is more than one data item, separate them with a space. |  
| ⚡ If a name consists of more than one words, use underscore (_) to represent the space, e.g. tree_nuts. |  
+-----+  
  
9. Allergens          : coffee  
Enter serial number for item 1: |
```

Figure 4.1.12 Sample output of function that validate only alphabetic characters

Figure above shows that the validation functions will prompt user again until user enter a valid input before proceeding to next stage. These functions streamline validation checks, ensuring consistency and reducing redundancy in input handling.

4.2 Main

```
22
23
24 ...
25
26 # print the bakery illustration and bakery name
27 print(bakery)
28
29 print('*****')
30 print('\t\tWELCOME TO (MORNING GLORY BAKERY)')
31 print('*****')
32
33 # provide the options for different roles
34 while True:
35     print('\nRole Options: ')
36     print('1. Manager 🍞')
37     print('2. Baker 🍞')
38     print('3. Cashier 💵')
39     print('4. Customer 🍞')
40     print('5. Exit the program 🚪')
41
42     # identify the role and run each role's function
43     try:
44         role = int(input('\nWho are you? (1, 2, 3, 4):\n>> '))
45
46         if role == 1:
47             manager.manager()
48
49         elif role == 2:
50
51         elif role == 3:
52
53         elif role == 4:
54
55         else:
56             print('Invalid input')
57
58     except ValueError:
59         print('Please enter a valid integer value')
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
279
280
281
282
283
284
285
286
287
288
289
289
290
291
292
293
294
295
296
297
298
299
299
300
301
302
303
304
305
306
307
308
309
309
310
311
312
313
314
315
316
317
318
319
319
320
321
322
323
324
325
326
327
328
329
329
330
331
332
333
334
335
336
337
338
339
339
340
341
342
343
344
345
346
347
348
349
349
350
351
352
353
354
355
356
357
358
359
359
360
361
362
363
364
365
366
367
368
369
369
370
371
372
373
374
375
376
377
378
379
379
380
381
382
383
384
385
386
387
388
389
389
390
391
392
393
394
395
396
397
398
399
399
400
401
402
403
404
405
406
407
408
409
409
410
411
412
413
414
415
416
417
418
419
419
420
421
422
423
424
425
426
427
428
429
429
430
431
432
433
434
435
436
437
438
439
439
440
441
442
443
444
445
446
447
448
449
449
450
451
452
453
454
455
456
457
458
459
459
460
461
462
463
464
465
466
467
468
469
469
470
471
472
473
474
475
476
477
478
479
479
480
481
482
483
484
485
486
487
488
489
489
490
491
492
493
494
495
496
497
498
499
499
500
501
502
503
504
505
506
507
508
509
509
510
511
512
513
514
515
516
517
518
519
519
520
521
522
523
524
525
526
527
528
529
529
530
531
532
533
534
535
536
537
538
539
539
540
541
542
543
544
545
546
547
548
549
549
550
551
552
553
554
555
556
557
558
559
559
560
561
562
563
564
565
566
567
568
569
569
570
571
572
573
574
575
576
577
578
579
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
598
599
599
600
601
602
603
604
605
606
607
608
609
609
610
611
612
613
614
615
616
617
618
619
619
620
621
622
623
624
625
626
627
628
629
629
630
631
632
633
634
635
636
637
638
639
639
640
641
642
643
644
645
646
647
648
649
649
650
651
652
653
654
655
656
657
658
659
659
660
661
662
663
664
665
666
667
668
669
669
670
671
672
673
674
675
676
677
678
679
679
680
681
682
683
684
685
686
687
688
689
689
690
691
692
693
694
695
696
697
698
698
699
699
700
701
702
703
704
705
706
707
708
709
709
710
711
712
713
714
715
716
717
718
719
719
720
721
722
723
724
725
726
727
728
729
729
730
731
732
733
734
735
736
737
738
739
739
740
741
742
743
744
745
746
747
748
749
749
750
751
752
753
754
755
756
757
758
759
759
760
761
762
763
764
765
766
767
768
769
769
770
771
772
773
774
775
776
777
778
779
779
780
781
782
783
784
785
786
787
788
789
789
790
791
792
793
794
795
796
797
797
798
799
799
800
801
802
803
804
805
806
807
808
809
809
810
811
812
813
814
815
816
817
818
819
819
820
821
822
823
824
825
826
827
828
829
829
830
831
832
833
834
835
836
837
838
839
839
840
841
842
843
844
845
846
847
848
849
849
850
851
852
853
854
855
856
857
858
859
859
860
861
862
863
864
865
866
867
868
869
869
870
871
872
873
874
875
876
877
878
879
879
880
881
882
883
884
885
886
887
888
889
889
890
891
892
893
894
895
896
897
897
898
899
899
900
901
902
903
904
905
906
907
908
909
909
910
911
912
913
914
915
916
917
918
919
919
920
921
922
923
924
925
926
927
928
929
929
930
931
932
933
934
935
936
937
938
939
939
940
941
942
943
944
945
946
947
948
949
949
950
951
952
953
954
955
956
957
958
959
959
960
961
962
963
964
965
966
967
968
969
969
970
971
972
973
974
975
976
977
978
979
979
980
981
982
983
984
985
986
987
987
988
989
989
990
991
992
993
994
995
996
997
998
999
999
1000
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1097
1098
1099
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1188
1189
1190
1191
1192
1193
1194
1195
1195
1196
1197
1198
1199
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1288
1289
1290
1291
1292
1293
1294
1295
1296
1296
1297
1298
1299
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1388
1389
1390
1391
1392
1393
1394
1395
1396
1396
1397
1398
1399
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1488
1489
1490
1491
1492
1493
1494
1495
1496
1496
1497
1498
1499
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1588
1589
1590
1591
1592
1593
1594
1595
1596
1596
1597
1598
1599
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1688
1689
1690
1691
1692
1693
1694
1695
1696
1696
1697
1698
1699
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1788
1789
1790
1791
1792
1793
1794
1795
1796
1796
1797
1798
1799
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1888
1889
1890
1891
1892
1893
1894
1895
1896
1896
1897
1898
1899
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1988
1989
1990
1991
1992
1993
1994
1995
1996
1996
1997
1998
1999
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2088
2089
2090
2091
2092
2093
2094
2095
2096
2096
2097
2098
2099
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2188
2189
2190
2191
2192
2193
2194
2195
2196
2196
2197
2198
2199
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2219
2220
2221
```

Sample output:

```
MORNING GLORY BAKERY
+-----+
|   |   |   |   |
|   |   |   |   |
|   |   |   |   |
|   |   |   |   |
+-----+ +-----+
|   |   |   |   |
|   |   |   |   |
|   |   |   |   |
|   |   |   |   |
+-----+ +-----+
*****
WELCOME TO <MORNING GLORY BAKERY>
*****
Role Options:
1. Manager 🧑‍🍳
2. Baker 🧑‍🍳
3. Cashier 💸
4. Customer 🍔
5. Exit the program ➕✖️
Who are you? (1, 2, 3, 4):
>>>
```

Figure 4.2.2 Sample output of main function

```
*****
WELCOME TO <MORNING GLORY BAKERY>
*****
Role Options:
1. Manager 🧑‍🍳
2. Baker 🧑‍🍳
3. Cashier 💸
4. Customer 🍔
5. Exit the program ➕✖️
Who are you? (1, 2, 3, 4):
>>> 6

+-----+
| ▲ Invalid input. |
+-----+

Role Options:
1. Manager 🧑‍🍳
2. Baker 🧑‍🍳
3. Cashier 💸
4. Customer 🍔
5. Exit the program ➕✖️
Who are you? (1, 2, 3, 4):
>>>
```

Figure 4.2.3 Sample output of main function with invalid input

The bakery illustration will be displayed followed by a welcome message. The list of role options is displayed for user to select their role. User will only need to input a number and

proceed to respective function of the system. A warning message will be displayed if the input is not an integer or out of the range, then it will show the role options and prompt user to enter again.

4.3 Manager

4.3.1 Sign up / login page

```
128 def manager():
129     print('')
130     printed_centered('MANAGER')
131     manager_name = input('Name: ')
132     if manager_name in manager_info:
133
134         manager_username = input('Username: ')
135         while manager_username != (manager_info[manager_name]['manager_username']):
136             print('\n+-----+')
137             print('|\u25bc Username doesn\'t match. Please enter again. |')
138             print('+-----+\n')
139             manager_username = input('Username: ')
140
141         manager_password = input('Password: ')
142         while manager_password != 'd0ugh8o5s': # check whether the manager password is correct
143             print('\n+-----+')
144             print('|\u25bc Incorrect password. Please enter again. |')
145             print('+-----+\n')
146             manager_password = input('Password: ')
147
148     else:
149         # prevent unauthorized user to access manager privilege
150         print('\n+-----+')
151         print('|\u25bc You are not a manager, cannot access to manager privilege. |')
152         print('+-----+\n')
153         print('If you are a manager but does not have an account, please proceed to sign up.')
154
```

Figure 4.3.1.1 Manager main function

The ‘MANAGER’ is printed centre to act as a banner that notify users that they are now in the manager function. The function will prompt users to enter their name followed by the username and password. The warning messages will be displayed and prompt manager to enter again if the input criteria don’t be met. If the name is not found in file, the system is assumed that the user is not a manager and will avoid him/her to access the manager privilege, the user will return to the main page. If the user is a manager but doesn’t not have an account, they will be directing to the account sign up page to fill out the information needed such as username, gender, contact number and email address. After the sign-up process, they can then access to the manager privilege for further action.

Sample output:

```
-----  
          MANAGER  
-----  
  
Name: tsuki  
Username: tsukiyaki  
Password: d0ugh8o5s  
  
Login successfully!  
Welcome, manager tsuki !  
  
-----  
          MANAGER PRIVILEGE  
-----  
  
a. System Administration  
b. Order Management  
c. Financial Management  
d. Inventory Control  
e. Customer Feedback  
f. Notifications  
g. Back to Main page  
  
Select a choice (a, b, c, d, e, f, g):  
>>> |
```

Figure 4.3.1.2 Sample output of a manager that already has an account

```
-----  
          MANAGER  
-----  
  
Name: benny  
  
+-----+  
| ▲ You are not a manager, cannot access to manager privilege. |  
+-----+  
  
If you are a manager but does not have an account, please proceed to sign up.  
Are you a manager? (y=yes, n=no)  
>>> y  
💡 Sorry, the manager position is full.  
  
Exiting to Main page.....  
  
Role Options:  
1. Manager 🏢👩  
2. Baker 🍞👩  
3. Cashier 💵👩  
4. Customer 🧑👩  
5. Exit the program ➔⬅  
  
Who are you? (1, 2, 3, 4):  
>>> |
```

Figure 4.3.1.3 Sample output of a manager that does not have an account

```

1   {
2     "tsuki": {
3       "manager_username": "tsukiyaki",
4       "manager_password": "d0ugh8o5s",
5       "age": 45,
6       "gender": "female",
7       "contact_no": "123-3452621",
8       "email": "tsuki857@gmail.com"
9     }
10  }

```

Figure 4.3.1.4 Manager data in text file

Figure 4.3.1.4 shows the information of the only manager in the bakery. Figure 4.3.1.2 shows how a manager that found in the file login to the system to access the privileges. Figure 4.3.1.3 shows a manager that doesn't find in the file, is prompted to navigate to the sign-up page if they are a manager. If there is already another information of manager that store in the file, he/she will be directed to the main page as manager can only has 1.

4.3.2 System Administration

```

14 def system_administration():
15
16     while True:
17         print('')
18         printed_centered('ROLE MANAGEMENT')
19         print('1. Baker 🍞\n2. Cashier 💰\n3. Customer 🌟\n4. Back to Manager Privilege 🔒') # provide role selection
20
21         role = input('\nWhich role do you want to manage(1, 2, 3, 4):\n>>> ') # identify which role to manage
22
23         if role not in ['1', '2', '3', '4']:
24             print('+' + '-----+')
25             print('|\t Invalid input. Please enter again. |')
26             print('+' + '-----+')
27
28         elif role == '1': # manage baker
29             system_administration_baker.system_administration_baker()
30
31         elif role == '2': # manage cashier
32             system_administration_cashier.system_administration_cashier()
33
34         elif role == '3': # manage customer
35             system_administration_customer.system_administration_customer()
36
37         elif role == '4': # return to the previous page
38             print('\nExiting to Manager Privilege.....')
39             break
40
41         return False
42

```

Figure 4.3.2.1 Manager system administration function

A while loop is used in this function to repeat the process if any unexpected input found, such as accidentally input incorrectly, or invalid input. The manager can select a role to manage by inputting a number based on the Role Management menu, and the respective function for each option will be called if it is selected. The option is invalid if it is out of range. Option 1 is administrating bakers, option 2 is administrating the cashier and option 3 is administrating customers. The ‘return False’ statement is used to stop the execution within the function and return to the Manager Privilege if option 4 is chosen.

Sample output:

```
-----  
          ROLE MANAGEMENT  
-----  
1. Baker 🍞🍞  
2. Cashier 💰🍞  
3. Customer 🍔🍔  
4. Back to Manager Privilege 🔴⬅  
  
Which role do you want to manage(1, 2, 3, 4):  
>>> XXX  
  
-----+  
|⚠ Invalid input. Please enter again. |  
+-----+  
  
-----  
          ROLE MANAGEMENT  
-----  
1. Baker 🍞🍞  
2. Cashier 💰🍞  
3. Customer 🍔🍔  
4. Back to Manager Privilege 🔴⬅  
  
Which role do you want to manage(1, 2, 3, 4):  
>>>
```

Figure 4.3.2.2 Sample output of system administration function with an invalid input attempt

The manager can select one of the roles to manage from the role management. ‘Invalid input’ warning message is shown and prompt the manager to enter again when the input exceeds the range given.

System Administration of Baker & System Administration of Cashier

(a) Add bakers or cashier

```
39 def baker_accounts():
40     baker = load_data_from_baker() # store the data that retrieved from file into baker variable
41
42     baker_username = input('Username: ') # ask for baker's username
43     while baker_username in (baker[baker_name]['baker_username'] for baker_name in baker): # continue looping if there is a du
44         print('\n+-----+')
45         print('!▲ Username has been used. Please enter another username. !')
46         print('+-----+')
47         baker_username = input('\nUsername: ')
48
49     while True:
50         baker_password = 'b@k3rm4st3r!' # set the password for cashier
51         print('\n+-----+')
52         print('! ? The default password for cashier is "b@k3rm4st3r!". You can change it later. !')
53         print('+-----+')
54         break
55
56     while True:
57         try:
58             age = int(input('Age: ')) # ask for baker's age
59             if age < 18 or age > 60: # check if the age is between 18 - 60
60                 print('\n+-----+')
61                 print('!▲ The required age is between 18 and 60. Please enter a valid age. !')
62                 print('+-----+')
63             else:
64                 break # exit the loop if age is valid
65         except ValueError:
66             print('\n+-----+')
67             print('!▲ Please enter a valid age. !')
```

Figure 4.3.2.3 Function to fill out the information to add a baker

```
40 def cashier_accounts():
41     cashier = load_data_from_cashier() # store the data that retrieved from file into cashier variable
42
43     cashier_username = input('Username: ') # ask for cashier's username
44     while cashier_username in (cashier[cashier_name]['cashier_username'] for cashier_name in
45                                 cashier): # continue looping if there is a duplication of username
46         print('\n+-----+')
47         print('!▲ Username has been used. Please enter another username. !')
48         print('+-----+')
49         cashier_username = input('\nUsername: ')
50
51     while True:
52         cashier_password = 'securec@sh$' # set the password for cashier
53         print('\n+-----+')
54         print('! ? The default password for cashier is "securec@sh$". You can change it later. !')
55         print('+-----+')
56         break
57
58     while True:
59         try:
60             age = int(input('Age: ')) # ask for cashier's age
61             if age < 18 or age > 60: # check if the age is between 18 - 60
62                 print('\n+-----+')
63                 print('!▲ The required age is between 18 and 60. Please enter a valid age. !')
64                 print('+-----+')
65             else:
66                 break # exit the loop if age is valid
67         except ValueError:
```

Figure 4.3.2.4 Function to fill out the information to add a cashier

```

118 def system_administration_baker():
119     baker = load_data_from_baker() # store the data that retrieved from file into baker
120
121     while True:
122         print('')
123         printed_centered('SERVICES')
124         print('1. Add Baker(s)\n2. Remove Baker(s)\n3. Update Baker(s)\n4. Back to Role Management') # provide the option
125
126         manage_baker = input('\nPlease choose a service:\n>>> ')
127
128         if manage_baker == '1': # add baker
129             if len(baker) == 2:
130                 print('\nThe baker position is currently full.')
131             else:
132                 while True:
133                     print('')
134                     printed_centered('NEW BAKER ENTRY FORM')
135                     baker_accounts() # fill up the details of baker to complete the registration
136                     baker = load_data_from_baker() # read the information of baker again from file (the data will not be
137
138                     while True:
139                         add_more = input('Continue to add? (y=yes, n=no)\n>>> ') # after one baker has been added, ask user
140                         if add_more == 'y':
141                             if len(baker) == 2:
142                                 print('\nThe baker position is currently full.')
143                             break

```

Figure 4.3.2.5 Function to add bakers

```

122 def system_administration_cashier():
123     cashier = load_data_from_cashier() # store the data that retrieved from file into cashier
124
125     while True:
126         print('')
127         printed_centered('SERVICES')
128         print(
129             '1. Add Cashier(s)\n2. Remove Cashier(s)\n3. Update Cashier(s)\n4. Back to Manager Privilege') # provide the option
130
131         manage_cashier = input('\nPlease choose a service:\n>>> ')
132
133         if manage_cashier == '1': # add cashier
134             if len(cashier) == 1:
135                 print('\nThe cashier position is currently full.')
136             else:
137                 while True:
138                     print('')
139                     printed_centered('NEW CASHIER ENTRY FORM')
140                     cashier_accounts() # fill up the details of cashier to complete the registration
141                     cashier = load_data_from_cashier() # read the information of cashier again from file (the data will not be
142
143                     while True:
144                         add_more = input('Continue to add? (y=yes, n=no)\n>>> ') # after one cashier has been added, ask user
145                         if add_more == 'y':
146                             if len(cashier) == 2:
147                                 print('\nThe cashier position is currently full.')
148                             break

```

Figure 4.3.2.6 Function to add cashier

The bakers' data and cashiers' data are read from files and store in a baker variable and cashier variable for easy data access in the following code. The manager can choose the services to manage bakers or cashier and if add bakers or a cashier is chosen, it will check if the bakers in file has exactly 2 while the cashier in file has exactly 1, then manager is not allowed to add more unless a baker or a cashier is removed by selecting option 2.

To add a baker or cashier, `baker_accounts()` and `cashier_accounts()` function are called. Figure 4.3.2.3 showcase the code for `baker_accounts()` function. Figure 4.3.2.4 showcase the code for `cashier_accounts()` function. Username, gender, contact details and email address are needed to add a baker or a cashier. Once completed filling out the information, it will be saved to baker file or cashier file. Every while loop function act as a reattempt if invalid input is detected. When the information is correctly inputted, it will break the loop. The exception handling is used to validate the age if it is not an integer.

After adding a baker or a cashier, the manager can continue to add or stop adding. The break statement used in the last row in figure 4.3.2.5 and figure 4.3.2.6 is function to stop the loop for continue adding and exit to the service page.

Sample output:

```
-----  
          SERVICES  
-----  
1. Add Baker(s)  
2. Remove Baker(s)  
3. Update Baker(s)  
4. Back to Role Management  
  
Please choose a service:  
>>> 1  
  
The baker position is currently full.
```

Figure 4.3.2.7 Refuse to add baker

```
-----  
          SERVICES  
-----  
1. Add Cashier(s)  
2. Remove Cashier(s)  
3. Update Cashier(s)  
4. Back to Manager Privilege  
  
Please choose a service:  
>>> 1  
  
The cashier position is currently full.
```

Figure 4.3.2.8 Refuse to add cashier

```

Please choose a service:
>>> 1

-----
          NEW BAKER ENTRY FORM
-----

Username: crustKing
Password: b@k3rm4st3r!
Age: 45
Gender(m=male, f=female): m
Contact number(xxx-xxxxxx): 345-0965783
Email: crustking@gmail.com

crustking is added.

Continue to add? (y=yes, n=no)
>>>

```

Figure 4.3.2.9 Adding baker by filling out the form

```

Please choose a service:
>>> 1

-----
          NEW CASHIER ENTRY FORM
-----

Username: verticca
Password: securec@sh12
Age: 56
Gender(m=male, f=female): f
Contact number(xxx-xxxxxx): 342-0974682
Email: verticca@gmail.com

verticca is added.

Continue to add? (y=yes, n=no)
>>>

```

Figure 4.3.2.10 Adding cashier by filling out the form

```

1   {
2     "christine": {
3       "baker_username": "christine",
4       "baker_password": "23456789",
5       "age": 23,
6       "gender": "female",
7       "contact_no": "345-3039225",
8       "email": "christine@gmail.com"
9     },
10    "crustking": {
11      "baker_username": "crustking",
12      "baker_password": "b@k3rm4st3r!",
13      "age": 45,
14      "gender": "male",
15      "contact_no": "874-8977921",
16      "email": "crustking@gmail.com"
17    }
18  }

```

Figure 4.3.2.11 Bakers' data in text file

```
{  
    "verticca": {  
        "cashier_username": "verticca",  
        "cashier_password": "securec@sh12",  
        "age": 34,  
        "gender": "female",  
        "contact_no": "123-4567382",  
        "email": "verticca@gmail.com"  
    }  
}
```

Figure 4.3.2.12 Cashier's data in text file

Figure 4.3.2.7 to figure 4.3.2.10 are the process of adding bakers or cashier. Figure 4.3.2.7 and figure 4.3.2.8 refuse to add a baker or a cashier because the positions are full as there are already 2 bakers and 1 cashier in the system. Figure 4.3.2.9 and figure 4.3.2.10 demonstrate the information that needed to complete the form to add a baker or a cashier and ask for continue adding. Figure 4.3.2.11 and figure 4.3.2.12 are bakers and cashier's data store in text files.

(b) Remove bakers or cashier

Figure 4.3.2.13 Function to remove bakers

```

121     def system_administration_cashier():
122
123         elif manage_cashier == '2': # remove cashier
124             if len(cashier) == 0:
125                 print('\n ! There is no cashier in the bakery.')
126                 continue
127             else:
128                 print('\n+-----+')
129                 print('! To ensure the daily normal operation, you cannot remove the last cashier in the list!\n'
130                     '| unless you want to replace with another cashier. |')
131                 print('+-----+\n')
132
133             while True:
134                 remove_more = input('Continue to remove? (y=yes, n=no)\n>>> ') # after one cashier has been added, ask user if they
135                 if remove_more == 'y':
136                     break
137
138                 elif remove_more == 'n':
139                     print('\nStop removing. Exiting to Services page.....')
140                     break
141
142                 else:
143                     print('\n+-----+')
144                     print('! Invalid input. Please enter again. !')
145                     print('+-----+\n')
146
147                 if remove_more == 'n':
148                     break
149
150
151             print('')
152             printed_centered('CASHIER LIST')
153             index = 1

```

Figure 4.3.2.14 Function to remove cashier

In the function of removing bakers, if there is only 1 baker left in the file, the system will display a message to warn the manager not to remove the last baker for daily normal operations of the bakery as there will be no one in charge of producing the products and break out of the loop to return to the service page. Otherwise, the system will display a baker list and ask manager which baker to remove. If manager's input is invalid, it will repeat again the prompt. At this point, the repetition is worked with the while loop.

In the function of removing cashier, since there is only 1 cashier recorded in the system, a warning will be displayed when the manager wants to remove the cashier. If the manager insists on proceeding, the cashier can be removed. However, if he/she wants to remove additional cashier, the system will prevent the action and return to the service page as there are no more cashiers in the bakery to remove.

Sample output:

```
-----  
          SERVICES  
-----  
1. Add Baker(s)  
2. Remove Baker(s)  
3. Update Baker(s)  
4. Back to Role Management  
  
Please choose a service:  
>>> 2  
  
-----  
          BAKER LIST  
-----  
1. christine  
2. crustking  
3. cancel  
  
Which baker do you want to remove? (or enter 3 to cancel)  
>>> 1  
  
christine is removed.  
  
Continue to remove? (y=yes, n=no)  
>>> y  
  
+---+  
| ⚠ To ensure the daily normal operation, you cannot remove the last baker in the list. |  
+---+
```

Figure 4.3.2.15 Sample output of remove bakers

```
},  
    "crustking": {  
        "baker_username": "crustking",  
        "baker_password": "b@k3rm4st3r!",  
        "age": 45,  
        "gender": "male",  
        "contact_no": "012-4673894",  
        "email": "crustking@gmail.com"  
    }  
}
```

Figure 4.3.2.16 Data in the file after a baker is removed

```

Please choose a service:
>>> 2

+-----+
| ⚠ To ensure the daily normal operation, you cannot remove the last cashier in the list|
| unless you want to replace with another cashier. |
+-----+

Continue to remove? (y=yes, n=no)
>>> y

-----+
          CASHIER LIST
-----+
1. verticca
2. cancel

Which cashier do you want to remove? (or enter 2 to cancel)
>>> 1

verticca removed.

```

Figure 4.3.2.17 Sample output of remove a cashier

Figure 4.3.2.15 and figure 4.3.2.17 are the process of removing bakers or cashier. When the manager removes a baker, the system will notify manager that who is removed, followed by prompting them to continue remove or cancel. If ‘cancel’ is selected, it will return to the service page. Figure 4.3.16 shows the result after removing a baker. On the contrary, when the manager wants to remove a cashier, the system will warn him/her that if the last cashier is removed, the daily operations of the bakery might be affected because no cashier to generate sales report and complete the transaction with customers unless the manager want to replace with another cashier.

(c) Update bakers or cashier

```
117 def system_administration_baker():
118
119     elif manage_baker == '3': # update baker
120         while True:
121             print('')
122             printed_centered('BAKER LIST')
123             index = 1
124             for baker_list_key in baker:
125                 print(f'{index}. {baker_list_key}')
126                 index += 1
127             print(f'{len(baker) + 1}. cancel')
128
129
130             try:
131                 index_of_baker_to_edit = int(input(f'\nWhich baker do you want to edit? (or enter {len(baker) + 1} to cancel)\n>> '))
132                 if index_of_baker_to_edit == len(baker) + 1:
133                     print('\nCancelling. Exiting to services page.....')
134                     break
135
136                 elif 1 <= index_of_baker_to_edit <= len(baker):
137                     selected_baker = list(baker.keys())[index_of_baker_to_edit - 1] # identify the selected baker to update
138                     while True:
139                         printed_centered(f'{selected_baker.upper()}'S DATA')
140
141                         for baker_data_key, baker_data_value in (baker[selected_baker].items()):
142                             print(f'{baker_data_key}: {baker_data_value}') # print the details of baker and replace the underscore with a space
143
144                             attribute_of_baker_data = input(f'\nWhich information do you want to update? (or enter "cancel")\n>> ')
145                             if attribute_of_baker_data in baker[selected_baker]: # check if the attribute inputted found in baker's data
146
147                                 while True:
```

Figure 4.3.2.18 Function to update information of bakers

```
121 def system_administration_cashier():
122
123     elif manage_cashier == '3': # update cashier
124         while True:
125             print('')
126             printed_centered('CASHIER LIST')
127             index = 1
128             for cashier_list_key in cashier:
129                 print(f'{index}. {cashier_list_key}')
130                 index += 1
131             print(f'{len(cashier)} + 1}. cancel')
132
133
134             try:
135                 index_of_cashier_to_edit = int(
136                     input(f'\nWhich cashier do you want to edit? (or enter {len(cashier)} + 1} to cancel)\n>>> '))
137             if index_of_cashier_to_edit == len(cashier) + 1:
138                 print('\nCancelling. Exiting to Services page.....')
139                 break
140
141             elif 1 <= index_of_cashier_to_edit <= len(cashier):
142                 selected_cashier = list(cashier.keys())[index_of_cashier_to_edit - 1] # identify the selected cashier to update
143                 while True:
144                     print('\n-----')
145                     print(f'\t\t\t{selected_cashier.upper()}' S DATA')
146                     print('-----')
147
148                     for cashier_data_key, cashier_data_value in (cashier[selected_cashier].items()):
149                         print(
150                             f'{cashier_data_key}: {cashier_data_value}') # print the details of cashier and replace the underscore
```

Figure 4.3.2.19 Function to update information of cashier

In the service of update bakers or cashier, after selecting the username, information of that username will be displayed, and manager is required to enter which information to update. If the new value of username in existing file, the manager is required to enter again to avoid

duplication. If the information to update not found in files, a message ‘Data not found.’ is printed. After each update process, the updated information will be saved to files, and the program continue to execute until ‘cancel’ is inputted. Input ‘cancel’ will redirect the manager back to the services page.

Sample output:

```
Please choose a service:  
>>> 3  
  
-----  
BAKER LIST  
--  
1. christine  
2. crustking  
3. cancel  
  
Which baker do you want to edit? (or enter 3 to cancel)  
>>> 2  
  
-----  
CRUSTKING'S DATA  
--  
baker_username: crustking  
baker_password: b@k3rm4st3r!  
age: 45  
gender: male  
contact_no: 345-0965783  
email: crustking@gmail.com  
  
Which information do you want to update? (or enter "cancel")  
>>> contact_no  
  
Enter new contact_no: 874-8977921  
  
contact_no of crustking is updated.
```

Figure 4.3.2.20 Sample output of updating the information of a baker

```
1  {  
2      "crustking": {  
3          "baker_username": "crustking",  
4          "baker_password": "b@k3rm4st3r!",  
5          "age": 45,  
6          "gender": "male",  
7          "contact_no": "874-8977921",  
8          "email": "crustking@gmail.com"  
9      }  
10 }
```

Figure 4.3.2.21 The contact number in text file is updated

```

Please choose a service:
>>> 3

-----
          CASHIER LIST
-----

1. verticca
2. cancel

Which cashier do you want to edit? (or enter 2 to cancel)
>>> 1

-----
          VERTICCA'S DATA
-----

cashier_username: verticca
cashier_password: securec@sh123
age: 34
gender: female
contact_no: 123-4567382
email: verticca@gmail.com

Which information do you want to update? (or enter "cancel")
>>> cashier_password

Enter new cashier_password: c@shm4st$r

cashier_password of verticca is updated.

```

Figure 4.3.2.22 Sample output of update the information of a cashier

```

{
    "verticca": {
        "cashier_username": "verticca",
        "cashier_password": "c@shm4st$r",
        "age": 34,
        "gender": "female",
        "contact_no": "123-4567382",
        "email": "verticca@gmail.com"
    }
}

```

Figure 4.3.2.23 The password in text file is updated

When the manager choose option 3, which is ‘update bakers’ or ‘update cashiers’, a list of baker or cashier is displayed for manager to select. The manager can only enter the information given, else a message ‘Data not found.’ is shown. Upon selecting the information to update and enter a new value, the new value will then be saved to the files. Figure 4.3.2.21 and figure 4.3.2.23 shows the result baker and cashier’s data after updating the information.

System administration of Customers

```
302     def system_administration_customer():
303
304         while True:
305             print('')
306             printed_centered('SERVICES')
307             print(
308                 '|1.. Activate Customer(s)|n|2.. Deactivate Customer(s)|n|3.. Update Customer(s)|n|4.. Terminate/Remove Customer Account(s)')
309
310             manage_customer = input('\nPlease choose a service:\n>>> ')
311
312             if manage_customer == '1': # activate customer
313                 activate_customer()
314
315             elif manage_customer == '2': # deactivate customer
316                 deactivate_customer()
317
318             elif manage_customer == '3': # update customer
319                 update_customer()
320
321             elif manage_customer == '4': # terminate/remove customer
322                 terminate_customer()
323
324             elif manage_customer == '5': # return to the previous page
325                 print('\nExiting to Role Management.....')
326                 system_administration.system_administration()
327                 break
```

Figure 4.3.2.24 Primary function to manager customers

To manage customer accounts, manager has the option to activate, deactivate, update, terminate customer accounts and back to role management. The ‘while True’ loop is used to repeat to execute the code inside it when the input is not in the given options. When option 5 is chosen, the break statement is used to stop the loop and allow the manager to return to the role management page.

(a) Activate customers

```
40 def activate_customer():
41     while True:
42         inactive_acc = [] # create a list to store the names of inactive customers
43         for customer_name in customer:
44             if customer[customer_name]['account_status'] == 'inactive': # check if the account_status of customer is inactive
45                 inactive_acc.append(customer_name) # append inactive customers into the list
46
47         if len(inactive_acc) == 0: # if the inactive_acc list is empty
48             print('\n\tNo account to activate.')
49             print('Back to Services page.....')
50             break
51         else:
52             print('')
53             printed_centered('INACTIVE ACCOUNT')
54
55         index = 1
56         for name in inactive_acc:
57             print(f'{index}. {name}')
58             index = index + 1
59
60         index_cancel = index
61         print(f'{index_cancel}. cancel')
62
63     try:
64         activate = int(input('\nWhich account do you want to activate?\n>> '))
65         if activate == index_cancel: # cancel the process
66             break
```

Figure 4.3.2.25 Function to activate customers

In this function, an empty list ‘inactive_acc’ is created to store all the inactive accounts in the customer file for the manager to activate them. It will display the names in the list for the manager to make decision on which account to activate. If all account is activated, this means that the ‘inactive_acc’ list is empty. It will display a message that notify the manager no account to activate and redirect he/she back to the service page. when an account has been activated, a message will also be displayed to inform the manager that the account activation is successfully done, and the status changed is save to the file. Additionally, there is an exception handling to handle the ValueError and IndexError if the input is not an integer or out of the range of the list. It is used to display a warning message instead of raising an error.

Sample output:

```
-----  
          SERVICES  
-----  
1. Activate Customer(s)  
2. Deactivate Customer(s)  
3. Update Customer(s)  
4. Terminate/Remove Customer Account(s)  
5. Back to Role Management  
  
Please choose a service:  
>>> 1  
  
-----  
          INACTIVE ACCOUNT  
-----  
1. eddie  
2. ashley  
3. cancel  
  
Which account do you want to activate?  
>>> 1  
  
eddie's account has been activated.
```

Figure 4.3.2.26 Process to activate an account

```
"eddie": {  
    "customer_username": "_eddie",  
    "customer_password": "Eddie12345",  
    "age": 25,  
    "gender": "male",  
    "contact_no": "019-8765432",  
    "email": "_eddie@gmail.com",  
    "address": "789 Eddie Lane, Villageburg",  
    "account_status": "active",  
    "loyalty_points": 0  
},
```

Figure 4.3.2.27 The account status is changed to active

```

-----  

      SERVICES  

-----  

1. Activate Customer(s)  

2. Deactivate Customer(s)  

3. Update Customer(s)  

4. Terminate/Remove Customer Account(s)  

5. Back to Role Management  

Please choose a service:  

>>> 1  

  ♦ No account to activate.  

Back to Services page.....  

-----  

      SERVICES  

-----  

1. Activate Customer(s)  

2. Deactivate Customer(s)  

3. Update Customer(s)  

4. Terminate/Remove Customer Account(s)  

5. Back to Role Management  

Please choose a service:  

>>>

```

Figure 4.3.2.28 Situation of no inactive account of customers

In figure 4.3.2.26, when the ‘Activate Customer(s)’ service is selected, a list of inactive account is shown. The manager can choose which account to activate or enter ‘cancel’ to return to the service page. Figure 4.3.2.27 shows the result of an account has been activated. Figure 4.3.2.28 demonstrates that if there is no inactive account, the manager is redirect to the service page to choose another service.

(b) Deactivate customers

```

88     def deactivate_customer():
89         while True:
90             active_acc = [] # create a list to store deactivate customer names
91             for customer_name in customer:
92                 if customer[customer_name]['account_status'] == 'active': # check if the account status of customer is active
93                     active_acc.append(customer_name) # append active customers into the list
94
95             if len(active_acc) == 0: # if the active_acc list is empty
96                 print('\n  ♦ No account to deactivate.')
97                 print('Back to Services page.....')
98                 break
99             else:
100                 print('')
101                 printed_centered('ACTIVE ACCOUNT')
102
103             index = 1
104             for name in active_acc:
105                 print(f'{index}. {name}')
106                 index = index + 1
107
108             index_cancel = index
109             print(f'{index_cancel}. cancel')
110
111             try:
112                 deactivate = int(input('In which account do you want to deactivate?\n>>> '))
113                 # choose which customer to deactivate

```

Figure 4.3.2.29 Function to deactivate customers

A list with a variable name ‘active_acc’ is created. For all customer accounts that are in active status will be stored in the list. The manager can select the index of which account to deactivate in the list provided or enter ‘cancel’ to abandon the process. len(active_acc) represent the number of accounts in the ‘active_acc’ list. If it is equal to 0, it means that no active account to for the manager to deactivate.

Sample output:

```
Please choose a service:  
>>> 2  
  
-----  
ACTIVE ACCOUNT  
-----  
1. Mark  
2. Jane  
3. anna  
4. andrew  
5. leo  
6. julia  
7. 9sam  
8. violet  
9. Haechan  
10. jen  
11. SOS  
12. abc  
13. lum  
14. nct  
15. chew  
16. yeti  
17. cancel  
  
Which account do you want to deactivate?  
>>> 5  
  
leo's account has been deactivated.
```

Figure 4.3.2.30 Process to deactivate customers

```
"leo": {  
    "customer_username": "leo_tiger",  
    "customer_password": "LeoTiger99",  
    "age": 29,  
    "gender": "male",  
    "contact_no": "016-5566778",  
    "email": "leo_tiger@gmail.com",  
    "address": "321 Leo Way, City Center",  
    "account_status": "inactive",  
    "loyalty_points": 1100  
},
```

Figure 4.3.2.31 Result of the account status of leo changed to inactive

```
-----  
ACTIVE ACCOUNT  
-----  
1. Mark  
2. Jane  
3. anna  
4. andrew  
5. julia  
6. 9sam  
7. violet  
8. Haechan  
9. jen  
10. SOS  
11. abc  
12. lum  
13. nct  
14. chew  
15. yeti  
16. cancel  
  
Which account do you want to deactivate?  
>>> xxx  
+-----+  
| ▲ Invalid input. Please enter again. |  
+-----+  
  
-----  
ACTIVE ACCOUNT
```

Figure 4.3.2.32 Situation when an input is invalid

When service 2 is selected, which is deactivate customers, a list of active accounts is shown. When an account is selected, it will be deactivated. Figure 4.3.2.31 shows the result of an account status is changed to inactive. Same goes to the activation function mentioned earlier, no account to deactivate if the ‘active_acc’ list is empty. If the manager enters a value that are not in the list, an invalid input message will be displayed and prompt the manager to enter again.

(c) Update customers

```
156     def update_customer():
157
158         while True:
159             print('')
160             print('CUSTOMER LIST')
161             index = 1
162             for customer_list in customer:
163                 print(f'{index}. {customer_list}')
164                 index += 1
165             print(f'{len(customer) + 1}. cancel')
166
167         try:
168             index_of_customer_to_edit = int(
169                 input(f'\nWhich customer do you want to edit? (or enter {len(customer) + 1} to cancel)\n>>> '))
170             if index_of_customer_to_edit == len(customer) + 1: # cancel the process
171                 print('\nCancelling. Exiting to Services page.....')
172                 break
173
174             elif 1 <= index_of_customer_to_edit <= len(customer):
175
176                 selected_customer = list(customer.keys())[index_of_customer_to_edit - 1] # determine the selected customer
177                 while True:
178                     print('')
179                     printed_centered(f'{selected_customer.upper()}'S DATA')
180
181                     for customer_data_key, customer_data_value in (customer[selected_customer].items()):
182                         # display all the details of customers except their account status
183                         if customer_data_key != 'account_status':
```

Figure 4.3.2.33 Function to update information of customers

When the manager wants to update information of customers, this function will be called. A list of all customers is displayed for the manager to choose. The ‘cancel’ option will always be at the last after all customers in the file have been listed down. This option can return the manager to the service page. The manager can choose a customer by entering the respective index. Since the index of a list is started with 0, therefore the value inputted by the manager must be subtracted by 1 to access the selected customer’s data. After that, the data of selected customer exclude the account status and loyalty points are shown. The new value of data of the selected customer will be updated in the file. When the input cannot be found in the found, it will display data not found or customer not found. Every new value inputted will be validated to ensure its accuracy and prevent duplication.

Sample output:

```
8. julia
9. 9sam
10. violet
11. Haechan
12. jen
13. SOS
14. abc
15. lum
16. nct
17. chew
18. yeti
19. cancel

Which customer do you want to edit? (or enter 19 to cancel)
>>> 10

-----
          VIOLET'S DATA
-----

customer_username: violet23
customer_password: Violet23pass
age: 26
gender: female
contact_no: 012-2233445
email: violet23@gmail.com
address: 321 Violet Lane, Riverside

Which information do you want to update? (or enter "cancel")
>>> |
```

Figure 4.3.2.34 Process to update information of customers

```
-----
          VIOLET'S DATA
-----
customer_username: violet23
customer_password: Violet23pass
age: 26
gender: female
contact_no: 012-2233445
email: violet23@gmail.com
address: 321 Violet Lane, Riverside

Which information do you want to update? (or enter "cancel")
>>> contact_no

Enter new contact_no: 018-4254302

contact_no of violet is updated.

-----
          VIOLET'S DATA
-----
customer_username: violet23
customer_password: Violet23pass
age: 26
gender: female
contact_no: 018-4254302
email: violet23@gmail.com
address: 321 Violet Lane, Riverside
```

Figure 4.3.2.35 An information of the selected customer is updated

In figure 4.3.2.34, the manager selects customer number 10 to update her information. In figure 4.3.2.35, the manager selects the contact number to edit and therefore, new value of the contact number is updated.

(d) Terminate customers

```
251     def terminate_customer():
252
253         while True:
254             print('')
255             index = 1
256             printed_centered('CUSTOMER LIST')
257             for key in customer:
258                 print(f'{index}. {key}')
259                 index += 1
260             print(f'{len(customer) + 1}. cancel')
261
262         try:
263             index_of_customer_to_terminate = int(input('\nWhich customer do you want to terminate? (or enter {len(customer) + 1}'))
264             if index_of_customer_to_terminate == len(customer) + 1:
265                 print('\nCancelling. Exiting to Services page.....')
266                 break
267
268             elif 1 <= index_of_customer_to_terminate <= len(customer):
269                 customer_to_terminate = list(customer.keys())[index_of_customer_to_terminate - 1] # identify the selected customer
270                 del customer[customer_to_terminate] # delete the selected customer
271                 save_info(customer)
272                 print(f'\n{customer_to_terminate} terminated.\n') # inform user about the customers has been terminated
273
274             while True:
275                 terminate_more = input('Continue to terminate? (y=yes, n=no)\n>>> ')
276                 if terminate_more not in ['y', 'n']:
277                     print('\n+-----+')
278                     print('|▲ Invalid input. Please enter again. |')
279                     print('+-----+\n')
```

Figure 4.3.2.36 Function to terminate customers

In this function, we initialize the index to 1 and look for the customers in file, to list them out starting from 1. After printing one customer, the index is increasing by 1 and continue to print the next customer. The ‘cancel’ option will be displayed at the last to provide an option for the manager to reverse a step. When the selected customer is in the list, he/she will be terminated. The manager can choose to terminate another account by entering ‘y’ that represent yes in the prompt of ‘Continue to terminate?’, while ‘n’ that represents no to stop terminating.

Sample output:

```
4. anna
5. andrew
6. ashley
7. leo
8. julia
9. 9sam
10. violet
11. Haechan
12. jen
13. SOS
14. abc
15. lum
16. nct
17. chew
18. yeti
19. cancel

Which customer do you want to terminate? (or enter 19 to cancel)
>>> 18

yeti terminated.

Continue to terminate? (y=yes, n=no)
>>> n

Stop terminating. Exiting to Services page.....
```

SERVICES

Figure 4.3.2.37 Process to terminate an account

```
CUSTOMER LIST
-----
1. Mark
2. Jane
3. eddie
4. enna
5. andrew
6. ashley
7. leo
8. julia
9. 9sam
10. violet
11. Haechan
12. jen
13. SOS
14. abc
15. lum
16. nct
17. chew
18. cancel

Which customer do you want to terminate? (or enter 18 to cancel)
>>> 18

Cancelling. Exiting to Services page.....
```

SERVICES

Figure 4.3.2.38 Cancel the termination process

In figure 4.3.2.37, the manager selects a customer to terminate and a message that indicates the selected customer successfully terminated is displayed. A prompt that asks the manager whether to continue terminate other customer accounts is displayed and the manager refused to continue terminate. In figure 4.3.2.38, the manager chooses the last option to cancel the process and back to the service page.

4.3.3 Order Management

```

32     def order_management():
33         while True:
34             print('\n', '\t'*13, 'ORDER DETAILS')
35             print('-' * 125)
36             header = ['Cart Id'] # create a list for headers and append the first item in the list
37
38             for value in order_list.values(): # access the values of order_list
39                 for sub_key, sub_value in value.items(): # access the subkey and sub value in the value of order_list
40                     header.append(sub_key.title().replace('_', ' ')) # append other details of orders into the header list and replace all underscore with space
41                     break
42
43             print(f'{header[1]:<19}{header[6]:<19}{header[2]:<28}{header[3]:<24}{header[4]:<26}{header[5]:<26}{header[6]}') # display the headers
44             print('-' * 125)
45
46             # display the details of order_list and specify the spaces between each other
47             for cart_id, order_details in order_list.items():
48                 print(f'{order_details["order_id"]:<19}{cart_id:<19}{order_details["username"]:<20}{order_details["items_ordered"]:[0]:<24}{float(order_de
49
50                 for items in order_details["items_ordered"][1:]: # for each item in the list of items_ordered
51                     print(f'":<19}":<19}":<20}{items:<24}":<26}"}') # display the items line by line and other details that not relevant will
52                 print('')
53             print('-'*125, '\n')
54
55             order_id_to_update = input('Enter Order ID to update order status (or enter "back" to exit):\n>>> ') # identify the order id to update
56             if order_id_to_update == 'back':
57                 print('\nExiting to Manager Privilege...')
58                 break # exit the entire loop
59
60             # check if the input exists in order_list
61             for cart_id, order_details in order_list.items():

```

Figure 4.3.3.1 Function of order management

The title ‘ORDER DETAILS’ is printed at the beginning followed by the header. The header is a list created to store the subkey of the order list. It is initialized with a value, which is ‘Cart Id’. `cart_id` is a key of the nested dictionary. Therefore, it does not have a name like ‘username’ for its value. By the initialization, ‘Cart Id’ becomes a name for `cart_id`, and the following code loops through the subkey and append them into the list. The header is printed out by specifying the spaces needed between each word to ensure a clear visualization.

Since the subkey, ‘`items_ordered`’, is a list in the nested dictionary, after printing out the first item in it, it loops through the list again to display the items line by line instead of all items in one line. The manager will be prompted to choose an order by inputting the order id to update the status, either payment completed or cancelled. If the updated status is same with the current status, a reminder message will be displayed to ask manager to enter again. He/she can also enter ‘back’ to quit this order management function, return to manager privilege. The order

status in the text file will be updated simultaneously when the manager makes changes during the program execution. The system will also output a message to notify the manager that the order status is successfully updated.

Sample output:

ORDER DETAILS					
Order Id	Cart Id	Username	Items Ordered	Total Price (RM)	Status
ORD001	5580946551	Mark0825	banana bread x 1 red velvet x 1	15.00	Order Placed
ORD002	5684950188	03jane_	gingerbread x 2 red velvet x 1	15.50	Order Placed
ORD003	1544631417	_eddie	banana bread x 1 apple pie x 2	14.50	Canceled
ORD004	4418942694	anna328_	gingerbread x 3	7.50	Order Placed
ORD005	7841554976	andrew_520	gingerbread x 1 red velvet x 2	23.50	Order Placed
ORD006	5548156914	rachel_lee	banana bread x 2	9.00	Order Placed
ORD007	9875428598	john_smith	banana bread x 3 apple pie x 1	18.50	Canceled
ORD008	4567891234	mary_jane	gingerbread x 2 banana bread x 1	9.50	Order Placed

Figure 4.3.3.2 A table of customers' order details

ORD044	4992374927	chew	cheezy bun x 2 gingerbread x 2	14.00	Payment Completed
ORD045	6685353618	yeti	cheezy bun x 2 red velvet x 3	41.00	Payment Completed
ORD046	6975692887	nct	apple pie x 3	15.00	Order Placed
ORD047	4291632822	nct	gingerbread x 2	4.50	Order Placed
ORD048	6583265832	nct	cheezy bun x 4	19.00	Order Placed
ORD049	9149754275	nct	cheezy bun x 5	23.75	Order Placed
ORD050	7498717867	nct	gingerbread x 5	11.25	Order Placed
ORD051	5739697651	nct	cheezy bun x 5	22.50	Order Placed
<hr/>					
Enter Order ID to update order status (or enter "back" to exit):					
>> ORD048					
Please update the status [payment completed, canceled (or enter "back" to return back)]:					
>> payment completed					
Status of order ID ORD048 is updated to "Payment Completed".					

Figure 4.3.3.3 Process of updating a status for an order

```

"6583265832": {
    "order_id": "ORD048",
    "username": "nct",
    "items_ordered": [
        "cheezy bun x 4"
    ],
    "total_price (RM)": "19.00",
    "order_date": "22-10-2024",
    "status": "Payment Completed"
},

```

Figure 4.3.3.4 Status of ORD048 changed to payment completed in the text file

A table-like structure is printed in figure 4.3.3.2 for the manager to monitor the order details and know which orders have not complete the payment. In figure 4.3.3.3, the manager successfully updates the order status of an order id to payment completed. Figure 4.3.3.4 shows the result of the order status of ORD048 has been changed to ‘Payment Completed’.

4.3.4 Financial Management

```

51 def financial_management():
52     months = ['January', 'February', 'March', 'April', 'May', 'June', 'July', 'August', 'September', 'October',
53               'November', 'December']
54     today_income = 0
55     today_expenses = 0
56     this_month_income = 0
57     total_income_so_far = 0
58
59     for receipt_details in transaction_keeping.values():
60         order_date = datetime.strptime(receipt_details['order_date'], '%d-%m-%Y') # retrieve all date from order_list
61         total_income_so_far += float(receipt_details['total_spend(RM)'])
62
63         if order_date.strftime('%d-%m-%Y') == datetime.now().strftime('%d-%m-%Y'):
64             today_income += float(receipt_details['total_spend(RM)'])
65
66         if order_date.month == datetime.now().month:
67             this_month_income += float(receipt_details['total_spend(RM)'])
68
69     for equipment_details in equipment_report_keeping.values():
70         repair_date = equipment_details['report_date']
71         unit, repair_cost = equipment_details['repair_cost'].split(' ')
72
73         if repair_date == datetime.now().strftime('%d-%m-%Y'):
74             today_expenses += float(repair_cost)
75
76     for ingredient_details in ingredient_keeping.values():
77         purchase_date = ingredient_details['purchase_date']
78         unit, ingredient_cost = ingredient_details['cost_per_unit'].split(' ')
79         cost = float(ingredient_cost) * ingredient_details['quantity_purchased']

```

Figure 4.3.4.1 Function of financial management

The name of months is stored in a list. The value current day's income and expenses, current month's income and total income so far are initialized to 0, ready for the calculation later. The code is using two types of datetime converter: strftime (%d-%m-%Y) and strptime (%d-%m-%Y). ‘strftime (%d-%m-%Y)’ is used to convert the date into a string with the format inside the parenthesis, while strptime (%d-%m-%Y) is used to convert back the date into datetime with the specific format too.

In the first for loop, it accesses all the receipt details in transaction keeping file, retrieve the order dates and total spend of customers, store them into variables. The order dates are converted from string format into datetime format, to easily obtain the months of order dates to compare with the current month. If the date equals to current date, today's income will be summed with the total_spend. On the other hand, if the month of date equals to current month, current month's income will be summed with the total_spend. The second and third for loop have the same concept with the first one, but they are used to calculate the expenses, including the cost regarding equipment and ingredients.

After all, the calculation results are displayed and follow by providing services for the manager to choose. The manager can track the income, expenses and profitability of the bakery by entering the desired year and month, the system will then show the outcomes filtered by the manager in a table. If the year or month entered cannot be found in the text file, a message ‘No record found’ is displayed and prompt the manager to reselect. For the profitability, there are 2 types of report, which are annual profit summary and monthly profit summary. The word ‘deficit’ will be shown for negative net profit, ‘surplus’ for positive net profit, and ‘break-even point’ for zero profit. The manager can also view the profit margin to understand the bakery performance in a month or in a year. The error handling of ZeroDivisionError is because the code snippets involve calculations that might be divided by zero. For example, during the calculation of profit margin, if total sales are zero, net profit divide total sales will give an error. Therefore, the error handling is used to display a warning message instead of raising an error.

Sample output:

```
⌚ Today's Income : RM 0.00 ⌚ October's Income So Far : RM 488.66
⌚ Today's Expenses : RM 0.00 ⌚ Total Income So Far : RM 536.36

-----
Financial Management
-----

1. Track Income
2. Track Expenses
3. Track Profitability
4. Back to Manager Privilege

Which financial data do you want to track:
>>> 1

Insert a year (or enter "0" to cancel):
>>> 2024

Insert the name of a month (or enter "cancel" to reselect year):
>>> september

$ 2024 September Income: RM 47.70
=====
Date           Description          Income(RM)
=====
19-09-2024     gingerbread x 6      15.90
19-09-2024     gingerbread x 7      18.55
```

Figure 4.3.4.2 Sample output of income tracking

```
"MGB-3424932": {
    "order_id": "ORD020",
    "items": [
        "gingerbread x 6"
    ],
    "order_date": "19-09-2024",
    "total_spend(RM)": 15.9
},
"MGB-6170747": {
    "order_id": "ORD021",
    "items": [
        "gingerbread x 7"
    ],
    "order_date": "19-09-2024",
    "total_spend(RM)": 18.55
},
"MGB-6407557": {
    "order_id": "ORD022",
    "items": [
        "gingerbread x 5"
    ],
    "order_date": "19-09-2024",
    "total_spend(RM)": 13.25
},
```

Figure 4.3.4.3 Customers' orders data that have completed the payment

```

Financial Management
-----
1. Track Income
2. Track Expenses
3. Track Profitability
4. Back to Manager Privilege

Which financial data do you want to track:
>>> 2

Insert a year (or enter "0" to cancel):
>>> 2023

Insert the name of a month (or enter "cancel" to reselect year):
>>> august

$ 2023 August Expenses: RM 944.00
=====
Date           Description           Expenses(RM)
=====
09-08-2023     Toaster oven      888.00
                stand mixer      56.00

Insert a year (or enter "0" to cancel):
>>> |

```

Figure 4.3.4.4 Sample output of expenses tracking

```

"6029": {
    "category": "Ovens and Baking Appliances",
    "equipment_name": "Toaster oven",
    "serial_number": "toasteroven1",
    "model_number": "TSTROVN563",
    "repair_cost": "RM 888.00",
    "manufacturer_email": "nctdream@no1.com",
    "warranty": "2 years",
    "report_date": "09-08-2023",
    "current_condition": "function well",
    "malfunction_date": "09-08-2023",
    "last_maintenance_date": "11-01-2023",
    "description": "dfbyfb"
},
"8604": {
    "category": "Mixing Equipment",
    "equipment_name": "stand mixer",
    "serial_number": "mixer1",
    "model_number": "STDMMXR034",
    "repair_cost": "RM 56.00",
    "manufacturer_email": "cheflum@super.com",
    "warranty": "1 year",
    "report_date": "09-08-2023",
    "current_condition": "waiting to replace with a new one",
    "malfunction_date": "09-08-2023",
    "last_maintenance_date": "26-04-2023",
    "description": "dfveyv"
},

```

Figure 4.3.4.5 Record keeping of repaired equipment

Figure 4.3.4.2 and figure 4.3.4.4 demonstrates how the manager track the income and expenses. Today's income and expenses, this month income, and total income so far are displayed at the

top when the program is executed. When the manager selects the year and the month of income or expenses that he/she wants to track, the output will be shown in a table. After each time output, the manager will be prompted to select another year until '0' is entered to cancel the process. Figure 4.3.4.3 and figure 4.3.4.5 shows the sample data of income and expenses that stored in text files.

```
Profit Tracking:  
1. Annual Profit Summary  
2. Monthly Profit Summary  
  
Type of profit to track (1, 2 or "cancel"): 1  
  
Insert a year (or enter "0" to cancel):  
>>> 2024  
  
-----  
[ 2024 PROFIT SUMMARY ]  
-----  
Total sales: RM 536.36 (30 orders)  
Total expenses: RM 1177.20  
  
-----  
Net profit: RM -640.84 (deficit)  
Profit margin: -119.48%  
  
The bakery lost approximately RM -1.19 for every RM earned.  
! It is a significant loss. Please pay immediate attention to this financial issues.  
-----
```

Figure 4.3.3.6 Sample output of annual profit summary

```
Profit Tracking:  
1. Annual Profit Summary  
2. Monthly Profit Summary  
  
Type of profit to track (1, 2 or "cancel"): 2  
  
Insert a year (or enter "0" to cancel):  
>>> 2024  
  
Insert the name of a month:  
>>> october  
  
-----  
 2024 October PROFIT SUMMARY  
-----  
Total monthly sales: RM 51.75 (4 orders)  
Total expenses: RM 428.20  
  
Net profit: RM -376.45 (deficit)  
Profit margin: -727.44%  
  
The bakery lost approximately RM -7.27 for every RM earned.  
! The bakery might be facing financial struggles. Try to reduce costs and  
adjust pricing to prevent continued losses.  
-----
```

Figure 4.3.4.7 Sample output of monthly profit summary

Figure 4.3.4.6 and figure 4.3.4.7 are the sample output of the profit summary. In the summary, total sales, total expenses, net profit and profit margin are displayed. Based on the data provided, it will also calculate the yield or loss for every earned. If the profit margin is in negative, the system will alert the manager to be aware of the financial loss issues.

4.3.5 Inventory Control

```
369 def inventory_control():
370     while True:
371         print('')
372         printed_centered('MAIN INVENTORY MANAGEMENT')
373         print('1. Ingredient Inventory\n2. Product Inventory\n3. Back to Manager Privilege')
374
375         # determine which inventory to manage and execute their corresponding functions
376         control = input('\nSelect the inventory that you want to manage(1, 2, 3, 4):\n>>> ')
377
378         if control == '1':
379             manager_inventory_ingredient.ingredient_management()
380
381         elif control == '2':
382             inventory_control_product()
383
384         elif control == '3':
385             print('\nExiting to Manager Privilege.....')
386             break
387
388         else:
389             print('-----+')
390             print('!▲ Invalid input. Please enter again. !')
391             print('+-----+')
```

Figure 4.3.5.1 Function of the main inventory control

Figure above demonstrates the code snippets of the main inventory control function. Manager can control both ingredient and product inventory or exit to the privilege page by typing in the corresponding index number. An invalid input message is popped out if the value entered is out of the range.

(a) Product Inventory

```
19 def inventory_control_product():
20     while True:
21         if len(product_inventory) == 0:
22             print('\n! The product inventory is empty. Please restock in time.') # output a warning message when the product in inventory is empty
23
24         # display the options of how to control the product in the inventory
25         print('')
26         printed_centered('PRODUCT MANAGEMENT')
27         print('1. Add products into inventory\n2. Remove products from inventory\n3. Update products in inventory\n4. Back to Main Inventory Management')
28         product_control = input('\nWhat action do you wish to perform? (1, 2, 3, 4)\n>>> ') # collect user preference
29
30         while True:
31
32             if product_control == '1': # add products into inventory
33                 print('')
34                 printed_centered('CURRENT PRODUCTS PRODUCED BY BAKERS')
35
36                 # retrieve the information of product name and quantity to display
37                 for key, value in product.items():
38                     product_name = value['product_name']
39                     quantity = len(value['serial_number']) # determine how many serial number for each product to represent the quantity of the product
40
41                     print(f'Product name: {product_name}<15> quantity: {quantity}')
42
43                 chosen_product = input('\nPlease enter product name to restock (or enter "cancel" to cancel)\n>>> ') # collect user's chosen product
44
45                 if chosen_product == 'cancel':
46                     print('\nCencelling. Exiting to Product Management page.....')
47                     break
48
49                 product_found = False
```

Figure 4.3.5.2 Function of product inventory control

In figure 4.3.5.2, `len(product inventory) == 0` means that the product inventory is empty. The manager can have the privilege to add, remove and update the products in the inventory. If manager choose ‘1’, it will display the current quantity of products produced by bakers that have not been added into the inventory. The for loop is used to get the product name and quantity from the file and store it into variables for easy access. The quantity of the products is based on how many serial numbers a product has. The system will ask the manager which products he/she wants to add the stock.

If ‘cancel’ is inputted, it will break the inner while loop and rerun the outer while loop. Otherwise, it will initialize `product_found` to be False (no product found). When looping through the file, if the product selected by the manager is detected, `product_found` is changed to True and the manager can select how many products to be added into the inventory. While adding the stock, the manager is responsible to add the price in the format of float numbers or integers and description to be displayed on the digital menu. Upon adding, the quantity of products produced by bakers is deducted from the baker record keeping file to prevent confusion for restock next time. If the selected product not found in the baker record keeping file, it will show a message of ‘Product not found.’ to the manager and prompt he/she to choose another product. The quantity of products is not enough when the number of stocks to add is more than the number of products produced by bakers. after adding the stock, the quantity of products in baker product keeping will be reduced.

When the manager chooses to remove products from the inventory, the system will display the product name and the available quantity to remove. If the quantity to remove by the manager is more than the available quantity, it is out of range. If the product in the inventory is removed until finish, it will show a message that tell the manager the products has finished and ask bakers to bake more.

When the manager wants to update the information of the product in the inventory, he/she can choose which data to update. Information such as product name cannot be duplicated and stocks to update cannot exceed the number of products produced by bakers. Besides, the new value of price must be a number. All changes made are saved to files.

Sample output:

```
-----  
PRODUCT MANAGEMENT  
-----  
1. Add products into inventory  
2. Remove products from inventory  
3. Update products in inventory  
4. Back to Main Inventory Management    
  
What action do you wish to perform? (1, 2, 3, 4)  
>>> 1  
  
-----  
CURRENT PRODUCTS PRODUCED BY BAKERS  
-----  
Product name: banana bread      quantity: 1  
Product name: apple pie          quantity: 1  
Product name: gingerbread       quantity: 2  
Product name: cheezy bun        quantity: 2  
Product name: tiramisu          quantity: 4  
Product name: apple cake        quantity: 4  
  
Please enter product name to restock (or enter "cancel" to cancel)  
>>> banana bread  
  
Number of banana bread to add: 1  
  
1 banana bread(s) is(are) added into the inventory.  
Current stock of banana bread in product inventory: 3
```

Figure 4.3.5.4 Sample output of adding stock of product

```
"001": {  
    "product_name": "banana bread",  
    "stock": 3,  
    "price": "RM 4.50",  
    "description": "Rich in vitamin A and perfect for healthy treat"  
}  
}
```

Figure 4.3.5.5 Stock of banana bread is added

```

-----  

      PRODUCT MANAGEMENT  

-----  

1. Add products into inventory  

2. Remove products from inventory  

3. Update products in inventory  

4. Back to Main Inventory Management 
```

What action do you wish to perform? (1, 2, 3, 4)
>>> 2

```

-----  

      PRODUCT INVENTORY  

-----  

tiramisu      : 4  

cheezy bun    : 2  

red velvet    : 2  

apple pie     : 1  

gingerbread   : 3  

banana bread  : 3
```

Which products do you want to reduce? (or enter "cancel" to cancel)
>>> tiramisu

Number of tiramisu to reduce: 1

1 tiramisu(s) is[are] reduced.

Figure 4.3.5.6 Sample output of removing stock of product

```
{
  "009": {
    "product_name": "tiramisu",
    "stock": 3,
    "price": "RM 8.50",
    "description": "Coffee-soaked ladyfingers with creamy mascarpone and blueberry toppings"
  },
}
```

Figure 4.3.5.7 Stock of tiramisu is reduced

```

-----  

      PRODUCT LIST  

-----  

1. Tiramisu  

2. Cheezy Bun  

3. Red Velvet  

4. Apple Pie  

5. Gingerbread  

6. Banana Bread  

7. cancel  

Enter the product's index number to update: 2  

-----  

      CHEEZY BUN  

-----  

product_name: cheezy bun  

stock: 2  

price: RM 5.00  

description: Extra gooey cheese for cheese aficionado  

Which information do you want to update? (or enter "cancel")  

>>> price  

Enter new price: 4.5  

Price is updated.

```

Figure 4.3.5.8 Sample output of updating information of product in the inventory

```

"004": {  

    "product_name": "cheezy bun",  

    "stock": 2,  

    "price": "RM 5.00",  

    "description": "Extra gooey cheese for cheese aficionado"  

},

```

Figure 4.3.5.9 Price of cheezy bun is updated

In figure 4.3.5.4, the manager select option 1 to add stocks. The current quantity of product produced by bakers is displayed and the manager selects which product he/she wants to add into the inventory. Once the stock is added, the system will notify the manager how many products are added into the inventory and the current stock of product after added. Figure 4.3.5.6 demonstrates the process of removing stock from the inventory. Product names and their quantities that currently have in the inventory is showed to allow the manager to know how many products are available to reduce. In figure 4.3.5.8, it prompts the manager to choose a product to update its information. in this case, the price of cheezy bun is updated from RM5.00 to RM4.50. The update will be saved into the file. Figure 4.3.5.5 shows the the stock of banana bread is added, figure 4.3.5.7 shows the stock of tiramisu is reduced and figure 4.3.5.9 shows the price of cheezy bun is updated.

(b) Ingredient Inventory

```
89 def ingredient_management():
90     while True:
91         print('\n-----')
92         print('\t\t\t\t\tINGREDIENT MANAGEMENT')
93         print('-----')
94         print(
95             '1. Add Ingredients\n2. Remove Ingredients\n3. Update Ingredients\n4. Back to Main Inventory Management')
96
97         option_product_management = input('\nWhat action do you wish to perform? (1, 2, 3, 4)\n>>> ')
98         if validation_empty_entries(option_product_management):
99             if option_product_management not in ['1', '2', '3', '4']:
100                 print('\n-----+')
101                 print('!▲ Please enter a valid number. |')
102                 print('+-----+')
103             else:
104                 if option_product_management == '1':
105                     add_ingredient()
106                 elif option_product_management == '2':
107                     remove_ingredient()
108                 elif option_product_management == '3':
109                     update_ingredient()
110                 elif option_product_management == '4':
111                     print('\nExiting to Main Inventory Management page...')
112                     break
113
```

Figure 4.3.5.10 Function to control the ingredients in the inventory

```
115 def ingredient_categories():
116     category = None
117     while True:
118         print('\n-----')
119         print('|\t\tMAIN CATEGORIES OF INGREDIENTS')
120         print('-----')
121         print('1. Flours and Grains')
122         print('2. Sweeteners')
123         print('3. Fats and Oils')
124         print('4. Dairy and Non-Dairy Products')
125         print('5. Leavening Agents')
126         print('6. Spices and Flavourings')
127         print('7. Fillings and Toppings')
128         print('8. Fruits and Vegetables')
129         print('9. Preservatives and Stabilizers')
130         print('10. Others')
131         print('11. Back to Ingredient Management page')
132
133         option_product_categories = input('\nPlease input the category:\n    \'\\n>> \'')
134
135         if validation_empty_entries(option_product_categories):
136             if option_product_categories not in ['1', '2', '3', '4', '5', '6', '7', '8', '9', '10', "11"]:
137                 print('\n-----')
138                 print('|\t▲ Please enter a valid number. |')
139                 print('-----')
140                 continue
141
142             if option_product_categories == '11':
```

Figure 4.3.5.11 Function to determine the categories of ingredients

In figure 4.3.5.10, the manager can choose to add, remove or update the ingredients. Each choice will trigger the respective function to run. If the manager chosen option 4, she will return to the main inventory control. The function in figure 4.3.5.11 will show 10 main categories of ingredients, used to ask the manager which categories of ingredients to be added, and return its value. The manager can only choose the option given else it is invalid.

(a) Add ingredients

```
178 def add_ingredient():
179     category = ingredient_categories()
180
181     ingredient_info = ['Ingredient Name', 'Ingredient Form', 'Batch Number', 'Unit Measurement', 'Total Quantity Purchased',
182                         'Purchase Date (DD-MM-YYYY)', 'Expiry Date (DD-MM-YYYY)', 'Supplier Name',
183                         'Supplier Contact Number (xxx-xxxxxx)', 'Cost Per Unit', 'Storage Requirements',
184                         'Allergen Information']
185
186     max_length = 0
187     for item in ingredient_info:
188         if len(item) > max_length:
189             max_length = len(item)
190
191     print('\n+-----+')
192     print('| ♦ Fill out the following fields to add a new ingredient to the inventory. |')
193     print('+-----+')
194
195     while True:
196         ingredient_name = input(f'1. {ingredient_info[0]}.ljust(max_length + 2)}: ')
197         if validation_empty_entries(ingredient_name):
198             if validation_alphabet_only(ingredient_name):
199                 break
200             else:
201                 print('\n+-----+')
202                 print('!▲ Please enter a valid ingredient name. (Cannot contain any digits and special characters.) !')
203                 print('+-----+\n')
204
205     while True:
206         ingredient_form_list = {
```

Figure 4.3.5.12 Function to add ingredients into the inventory

This add_ingredient function is including the function in figure 4.3.5.12 to allow the manager to select ingredient categories at the beginning. ingredient_info is a list stores the information of ingredients for easy access in different function so that the programmers does not need to type the information each time they want to use. They can access the information by list indexing. The max_length is used to identify the maximum length among the items in ingredient_info list, to determine the position of the colon. This can enhance the overall visual effect to look not so messy. To add an ingredient into the inventory, the manager is required to fill out all the information. all entries must not be empty and will be validated to ensure its accuracy. For example, ingredient name cannot contain alphabets only. Each category has its specific form and unit measurement. The manager can only enter the allowable form or unit measurement. After adding an ingredient, the manager can continue to add another ingredient or exit to the ingredient management page.

Sample output:

```
+-----+
| ? Fill out the following fields to add a new ingredient to the inventory. |
+-----+
1. Ingredient Name : almond flour

? Allowable ingredient form: powdered, granules, rolled. ?
2. Ingredient Form : powdered
3. Batch Number : B009

? Allowable unit measurement: g, kg ?
4. Unit Measurement : kg
5. Cost Per Unit : 2
6. Total Quantity Purchased : 10
6. Purchase Date (DD-MM-YYYY) : 23-08-2024
7. Expiry Date (DD-MM-YYYY) : 23-08-2025
8. Supplier Name : Bob
9. Supplier Contact Number (xxx-xxxxxx) : 018-5730283

? Allowable storage requirement: dry storage, refrigerated, freezer ?
11. Storage Requirements : dry storage

+-----+
| ? If there is more than one data item, separate them with a space.           |
| ? If a name consists of more than one words, use underscore (_) to represent the space, e.g. tree_nuts. |
+-----+
12. Allergen Information : almond

Information saved. Continue adding? (y=yes, n=no)
>>> n
```

Figure 4.3.5.13 Filling out the information needed to add an ingredient into the inventory

```
"B009": {
    "category": "Flours and Grains",
    "ingredient_name": "almond flour",
    "ingredient_form": "powdered",
    "batch_number": "B009",
    "unit_measurement": "kg",
    "quantity_purchased": 10.0,
    "purchase_date": "23-08-2024",
    "expiry_date": "23-08-2025",
    "supplier_name": "Bob",
    "supplier_contact": "018-5730283",
    "cost_per_unit": "RM 2.00",
    "storage_requirement": "dry storage",
    "allergen_info": [
        "almond"
    ]
}
```

Figure 4.3.5.14 Ingredients that successfully added are saved to text file

Figure above shows the process of filling all the important information when adding an ingredient. Some information such as ingredient forms, unit measurements, and storage requirement are given hints to know what can be filled in. After all information has been filled,

figure 4.5.3.14 shows that it is saved to file and ask the manager whether she wants to continue adding or not.

(b) Remove ingredients

```
163     def remove_ingredient():
164         while True:
165             print('')
166             printed_centered('INGREDIENT LIST')
167             index = 1
168             for ingredient in ingredient_data.values():
169                 printf(f'{index}. {ingredient["ingredient_name"]}')
170                 index += 1
171             print(f'{len(ingredient_data) + 1}. cancel')
172
173             try:
174                 index_of_ingredient_to_remove = int(input(f'\nWhich ingredient do you want to remove? (or enter {len(ingredient_data) + 1} to cancel)\n>>> '))
175                 if index_of_ingredient_to_remove == len(ingredient_data) + 1: # cancel the process
176                     print('\nCancelling. Exiting to Ingredient Management page.....')
177                     break
178
179                 elif 1 <= index_of_ingredient_to_remove <= len(ingredient_data):
180                     ingredient_to_remove = list(ingredient_data.keys())[index_of_ingredient_to_remove - 1] # identify baker to remove by accessing the index of key of baker
181                     while True:
182                         print(f'\n💡 Current quantity of {ingredient_data[ingredient_to_remove]}["ingredient_name"].title(): {ingredient_data[ingredient_to_remove]}["quant')
183                         try:
184                             quantity_remove = float(input('Enter the quantity to remove: '))
185
186                             ingredient_data[ingredient_to_remove]['quantity_purchased'] -= quantity_remove # delete the selected baker
187                             print(f'\n{quantity_remove} {ingredient_data[ingredient_to_remove]["unit_measurement"]} of {ingredient_data[ingredient_to_remove]}["ingredient_')
188
189                             save_info(ingredient_data)
190                         break
```

Figure 4.3.5.15 Function to remove ingredients from the inventory

In this function, index is initialized to 1. A list of ingredients in the inventory is printed out with the increment of index. If manager input the option that is in the list, the current quantity of that ingredient is shown, and the manager can decide how many ingredients to remove. She can choose to continue removing or stop removing. For every incorrect input, the while loop is triggered and prompt the manager to enter again until the entry is accepted.

Sample output:

```
-----  
          INGREDIENT LIST  
-----  
1. whole wheat flour  
2. cornmeal  
3. maple syrup  
4. whipping cream  
5. candy  
6. apple  
7. chilly  
8. baking powder  
9. almond flour  
10. cancel  
  
Which ingredient do you want to remove? (or enter 10 to cancel)  
>>> 7  
  
💡 Current quantity of Chilly: 2.0 kg  
Enter the quantity to remove: 0.5  
  
0.5 kg of Chilly is removed.  
  
Continue to remove? (y=yes, n=no)  
>>> n  
  
Stop removing. Exiting to Ingredient Management page.....
```

Figure 4.3.5.16 Sample output of removing an ingredient

An ingredient list is displayed for the manager to select which ingredient to removed. In this case, 0.5kg of the seventh ingredient, chilly, is removed. The system then asks the manager to continue removing and she prefer to stop removing.

(c) Update information of ingredients

```
523 def update_ingredient():
524     while True:
525         index = 1
526         print('\n-----')
527         print('\t\t\t\t\tINGREDIENT LIST')
528         print('-----')
529         for key, value in ingredient_data.items():
530             print(f'{index}. {value["ingredient_name"].title()}')
531             index += 1
532         print(f'{len(ingredient_data) + 1}. cancel')
533
534     try:
535         ingredient_to_update = int(input('\nEnter the ingredient\'s index number to update: '))
536         if ingredient_to_update == len(ingredient_data) + 1: # cancel update
537             print('\nCancelling. Exiting to Ingredient Management page.....')
538             break
539
540         elif 1 <= ingredient_to_update <= len(ingredient_data):
541             selected_ingredient = list(ingredient_data.keys())[ingredient_to_update - 1]
542             while True:
543                 print('')
544                 printed_centered(ingredient_data[selected_ingredient]["ingredient_name"].upper())
545
546                 # display the details of ingredient in a custom format
547                 for ingredient_data_key, ingredient_data_value in (ingredient_data[selected_ingredient].items()):
548                     if ingredient_data_key == 'allergen_info':
549                         print(f'{ingredient_data_key}:: {", ".join(ingredient_data_value)}')
550                     else:
551                         print(f'{ingredient_data_key}:: {ingredient_data_value}'
```

Figure 4.3.5.17 Function of update the information of ingredients

A list of ingredient names will also be displayed to enable the manager to select the information to update. The allergen_info in the file is kept as a list, therefore “,”.join is used to rearrange the format of allergen_info and print out. Other information will be printed out as usual. When the manager selects the information to update, the new value will undergo a series of validation just like adding the ingredients. The information is saved to file after the update. An invalid input can cause the system to prompt the manager to enter again.

Sample output:

```
8. Baking Powder
9. Almond Flour
10. cancel

Enter the ingredient's index number to update: 5

-----
          CANDY
-----
category      : Sweeteners
ingredient_name : candy
ingredient_form  : granulated
batch_number   : B005
unit_measurement : g
quantity_purchased : 200.0
purchase_date   : 01-10-2024
expiry_date     : 08-08-2025
supplier_name   : charplin
supplier_contact : 234-3478532
cost_per_unit    : RM 0.50
storage_requirement : dry storage
allergen_info    : no

Which information do you want to update? (or enter "cancel")
>>> purchase_date

Enter new purchase_date: 25-10-2024

purchase_date of candy is updated.
```

Figure 4.3.5.18 Sample output of update information of an ingredient

```
"B005": {
    "category": "Sweeteners",
    "ingredient_name": "candy",
    "ingredient_form": "granulated",
    "batch_number": "B005",
    "unit_measurement": "g",
    "quantity_purchased": 200.0,
    "purchase_date": "25-10-2024",
    "expiry_date": "08-08-2025",
    "supplier_name": "charplin",
    "supplier_contact": "234-3478532",
    "cost_per_unit": "RM 0.50",
    "storage_requirement": "dry storage",
    "allergen_info": [
        "no"
    ]
},
```

Figure 4.3.5.19 Information of the selected ingredient is updated

When the manager chooses to update an ingredient, a list of ingredients is displayed. In this case, the manager selects the fifth ingredient which is candy, to update its purchase date. Upon the purchase date is successfully updated, a message appears to notify the manager about the update. Figure 4.3.5.19 shows the data in the text file about the purchase date of ‘candy’ is updated from 01-10-2024 to 25-10-2024.

4.3.6 Customer Feedback

```
41 def customer_feedback():
42     while True:
43         rating_five = 0
44         rating_four = 0
45         rating_three = 0
46         rating_two = 0
47         rating_one = 0
48         for review_details in review.values():
49             if review_details['rating'] == 5:
50                 rating_five += 1
51             elif review_details['rating'] == 4:
52                 rating_four += 1
53             elif review_details['rating'] == 3:
54                 rating_three += 1
55             elif review_details['rating'] == 2:
56                 rating_two += 1
57             else:
58                 rating_one += 1
59
60             print('')
61             printed_centered('CUSTOMER RATINGS OVERVIEW')
62             print(f'★★★★★{"5-star rating:".rjust(15)} {rating_five}')
63             print(f'★★★★{"4-star rating:".rjust(17)} {rating_four}')
64             print(f'★★★{"3-star rating:".rjust(19)} {rating_three}')
65             print(f'★★{"2-star rating:".rjust(21)} {rating_two}')
66             print(f'★ {"1-star rating:".rjust(23)} {rating_one}')
67
68         choice = input('\nDo you want to view and reply to customer feedback? (y=yes, n=no)\n>>> ')
```

Figure 4.3.6.1 Function about customer feedback

This function enables the manager to view the ratings and feedback of customers and reply to them. All codes are embedded in an infinite loop to ensure it can keep repeating the entire function when the inner loop is break, until the manager enter ‘n’ which is represent ‘no’ to break the loop. Initially, 5 variables regarding the ratings are set to 0, act as a counter. The for loop will loop through all data in the file and filter out the ratings, ultimately conduct additions to the variables set early. For example, if the customer ratings in the file is 5, the counter of rating_five will increase by 1, to total up how many customers rate the bakery with 5 stars. After the calculation, the result will be printed out and ask the manager if she decide to view and reply to the customer feedback. If yes, a table of the customer feedback details will be shown. the manager can enter a review id to reply to the customer feedback. If the entry does not find in the file, the manager is prompted to enter again. After the manager enter the reply text, the system will generate a random reply number and inform the manager that the reply has been sent. A copy of the reply is saved to the file. The manager can choose to reply to another customer or exit to the ratings overview.

Sample output:

```
CUSTOMER RATINGS OVERVIEW
★★★★★ 5-star rating: 5
★★★★ 4-star rating: 4
★★★ 3-star rating: 1
★★ 2-star rating: 0
★ 1-star rating: 0

Do you want to view and reply to customer feedback? (y=yes, n=no)
>>> y

CUSTOMER FEEDBACK
-----

| Review ID | Username   | Product Name                        | Review                                                                             | Rating |
|-----------|------------|-------------------------------------|------------------------------------------------------------------------------------|--------|
| 001       | anna328_   | gingerbread x 1                     | I love the flavor and the taste is excellent. Will definitely buy it again.        | 5      |
| 002       | rechel_lee | banana bread x 1                    | The banana bread was fresh and delicious.                                          | 4      |
| 003       | mary_jane  | gingerbread x 1<br>banana bread x 1 | Both the gingerbread and banana bread were wonderful! The flavors were spot on.    | 5      |
| 004       | jessica_k  | red velvet x 1<br>gingerbread x 1   | The red velvet was moist and tasty, but the gingerbread tasted a little bit weird. | 3      |
| 005       | john_smith | banana bread x 1                    | The banana bread was delicious, but the apple pie was a bit too sweet.             | 4      |


```

Figure 4.3.6.2 Sample output of customer feedback

```
-----

|     |            |                                     |                                                                                  |   |
|-----|------------|-------------------------------------|----------------------------------------------------------------------------------|---|
| 005 | john_smith | banana bread x 1<br>apple pie x 1   | The banana bread was delicious, but the apple pie was a bit too sweet.           | 4 |
| 006 | chris_d    | apple pie x 1<br>gingerbread x 1    | The apple pie was sweet and the gingerbread was nicely spiced. Great experience! | 5 |
| 007 | mike_h     | gingerbread x 1<br>banana bread x 1 | Loved the gingerbread and banana bread! Very nice texture and tasty.             | 4 |
| 008 | nct        | cheezy bun                          | nice                                                                             | 4 |
| 009 | Mark0825   | red velvet                          | good                                                                             | 5 |
| 010 | nct        | apple pie                           | good                                                                             | 5 |


-----
Enter the review id to respond the feedback (or enter "cancel" to return back):
>>> 008

Enter response text:
>>> Thank you for your support!

Generating reply number...

Reply has been sent!
Continue to reply customers? (y=yes, n=no)
>>>
```

Figure 4.3.6.3 Sample output of replying customer feedback

```
"REPLY-1012": {
    "username": "nct",
    "products": "cheezy bun",
    "review": "nice",
    "rating": 4,
    "reply": "Thank you for your support!"
}
```

Figure 4.3.6.4 Reply of the manager is saved to the text file

Figure 4.3.6.2 shows the customer ratings overview to the manager. The manager decides to view and reply to the customer feedback by typing ‘y’ that represent yes. A table of the feedback details is printed out to allow the manager to select feedback to reply. In figure 4.3.6.3, the manager selects the review id 008 and reply with a sentence: ‘Thank you for your support!’. The system generated a random reply number and informed the manager that the reply has been sent. She can continue to reply or return to the customer ratings overview. Figure 4.3.6.4 shows that the reply of manager towards username ‘nct’ is recorded into the text file.

4.3.7 Notifications

```

136     def notification():
137         while True:
138             notice = load_data_from_notification() # store the data that retrieved from file into notice
139             notifications = [] # create a list for the number of notifications received
140             print('')
141             printed_centered('NOTIFICATIONS')
142             # initialize the number of malfunction and maintenance reports to 0
143             malfunction_report = 0
144             maintenance_report = 0
145
146             # if there is any malfunction equipments or equipments that need maintenance
147             for equipment in notice.values():
148                 if equipment['current_condition'] == 'malfunction':
149                     malfunction_report += 1 # add the number of equipments that are malfunction to the report
150                 elif equipment['current_condition'] == 'maintenance needed':
151                     maintenance_report += 1 # add the number of equipments that need maintenance to the report
152
153             # display how many reports if the report is not empty
154             if malfunction_report != 0:
155                 print(f'⚠ {malfunction_report} notification(s) from Malfunction Report.')
156             if maintenance_report != 0:
157                 print(f'⚠ {maintenance_report} notification(s) from Maintenance Report.\n')
158             elif malfunction_report == 0 and maintenance_report == 0:
159                 print('🎉 Hooray! No notifications yet!\n') # if there is no report, print the message of no notification
160
161             print('\n1. Malfunction Report\n2. Maintenance Report\n3. Back to Manager Privilege')
162             choice_of_report = input('\nEnter a number to get more insights: ') # choose which report insights to see
163
164             if choice_of_report == '1': # malfunction report

```

Figure 4.3.7.1 Function of manager’s notifications

The code snippet in figure 4.3.7.1 created a list, named notifications, to store all the notifications received. The malfunction_report and maintenance_report are initialized to 0. When the manager receives notifications, the for loop is works to loop through the notifications. if it is about equipment malfunction, the malfunction_report will increase by 1, and same goes to maintenance report. At this point, all the notifications are categorized. the system will display the number of notifications of each report and prompt the manager to select which report she wants to get more insights. When malfunction report is chosen, a malfunction_equipment list is created to store the malfunction equipment names.

All notifications about malfunction equipment based on the malfunction_equipment list are shown. The manager will be prompted to enter an equipment serial number to settle the

malfunction problem. She can choose how she would like to repair the chosen equipment, either repair by herself, the bakers, or the manufacturer. If the manager chooses to bring it for the manufacturer to repair, a message of request the manager to reach out to the respective manufacturer by the email address given is printed out. Assumed that the equipment is repaired, the manager is asked for the condition of the equipment. She can only claim the warranty if the equipment is repaired by the manufacturer, no matter it is function well or not. To keep a record of the equipment repaired, the manager requires to provide the cost incurred and all details are saved to equipment record keeping file. The notifications about this equipment will then be deleted from notification file. The number of reports in malfunction_report is also decrease by 1. The equipment name is removed from the malfunction_equipment list.

On the other hand, if the maintenance report is chosen, a maintenance_equipment list is created to store the equipment names that need for maintenance. All notifications about equipment that need for maintenance based on the maintenance_equipment list are shown. The manager also needs to enter the serial number of equipment to settle the problem. The email of the manufacturer will be provided for manager to book a maintenance date. Assumed that the equipment is after maintenance, the manager needs to decide whether it is function well or not. If not, the manager can claim the warranty from the manufacturer as the equipment is broken during maintenance. For the maintenance record keeping, she also needs to provide the cost incurred by any additional cost such as a deductible. The notifications about this equipment will be deleted from the notification file and the number of reports in the maintenance_report list is also decrease by 1. The equipment name is removed from the maintenance_equipment list.

Sample output:

```
NOTIFICATIONS
-----
⚠ 1 notification(s) from Malfunction Report.
⚠ 2 notification(s) from Maintenance Report.

1. Malfunction Report
2. Maintenance Report
3. Back to Manager Privilege

Enter a number to get more insights: 1

-----
MALFUNCTION REPORT
-----
>Toaster Oven
Category: Ovens And Baking Appliances
Serial number: toasteroven1
Model number: TSTRDVN563
Report date: 15-09-2024
Current condition: malfunction
Malfunction date: 14-09-2024
Last maintenance date: 12-12-2023
Description: Heating element burnt out.

Enter serial number that need for repairment (or enter "done" to return back): toasteroven1
Do you want to repair yourself or contact the manufacturer? (r=repair yourself, c=contact manufacturer)
>>> r
```

Figure 4.3.7.2 Sample output that shows the malfunction equipment

```
{
    "toasteroven1": {
        "report_number": "REPORT-5744",
        "category": "Ovens and Baking Appliances",
        "equipment_name": "Toaster oven",
        "serial_number": "toasteroven1",
        "model_number": "TSTRDVN563",
        "manufacturer_email": "nctdream@no1.com",
        "warranty": "2 years",
        "report_date": "15-09-2024",
        "current_condition": "malfunction",
        "malfunction_date": "14-09-2024",
        "last_maintenance_date": "12-12-2023",
        "description": "Heating element burnt out."
    },
}
```

Figure 4.3.7.3 Data of malfunction equipment

```

-----  

MAINTENANCE REPORT  

-----  

↑ Stand Mixer ↑  

Category: Mixing Equipment  

Serial number: mixer1  

Model number: STDMXR034  

Manufacturer email: chefium@super.com  

Warranty: 1 year  

Report date: 15-10-2024  

Current condition: maintenance needed  

Severity: high  

Maintenance needed date: 15-10-2024  

Last maintenance date: 26-12-2023  

Next scheduled maintenance: 26-12-2024  

Description: The speed settings are unresponsive.  

↑ Toaster Oven ↑  

Category: Ovens And Baking Appliances  

Serial number: toasteroven2  

Model number: TSTR0VN563  

Manufacturer email: nctdream@no1.com  

Warranty: 2 years  

Report date: 15-10-2024  

Current condition: maintenance needed  

Severity: high  

Maintenance needed date: 15-10-2024  

Last maintenance date: 11-11-2023  

Next scheduled maintenance: 11-11-2024  

Description: The motor is making loud noise.  

-----  

Enter serial number that need for maintenance (or enter "done" to return back): mixer1  

Has the equipment function well? (y=yes, n=no)  

>>> n  

💡 There is a "1 year" warranty for this equipment.  

The equipment is probably broken. Please claim the warranty from the manufacturer.  

Please provide the cost incurred for this repair: RM 20  

Exiting to Notification page.....
```

Figure 4.3.7.4 Sample output of settling the maintenance report

Figure 4.3.7.2 shows how many notifications of malfunction report and maintenance report have. The manager chooses to settle the malfunction report and decide to repair herself. The equipment is function well after the repair; no cost incurred. In figure 4.3.7.4, the manager selects the maintenance report. The stand mixer is chosen for the maintenance, but it was not function well. The system display warranty that can be claimed for the equipment. There is also an additional cost incurred during the maintenance, which is RM20.

4.4 Customer

4.4.1 Customer Account Management

```
346     def customer_menu():
347         logged_in_username = None # Initialize as None, means that no user is logged in yet
348
349         while True:
350             print('\n-----')
351             print('\t\t\t\t', 'CUSTOMER')
352             print('-----')
353             print('\n+-----+')
354             print('| ♦ Please "Sign Up" if you\'re logging in the first time. |')
355             print('| Please "Login" if you already have an account. |')
356             print('+-----+\n')
357
358             print('\n-----')
359             print("WELCOME TO MORNING GLORY BAKERY!")
360             print('-----')
361             print("1. Sign Up")
362             print("2. Login")
363             print("3. Browse Products")
364             print("4. Shopping Cart")
365             print("5. Order Tracking")
366             print("6. Submit Review")
367             print("7. View Loyalty Rewards")
368             print("8. Update Personal Information")
369             print("9. Manage Account")
370             print("0. Exit")
371
372             option = input("Please select an option (0-9): ")
373
374             if option == "1":
375                 sign_up()
```

Figure 4.4.1.1 Function of Customer Menu

```

35 def sign_up():
36     customer_name = input("Name: ")
37     customer_username = input("Username: ")
38
39     if customer_username in [info['customer_username'] for info in customer_info.values()]:
40         print('\n+-----+')
41         print('!▲ Warning: One person can only have one account! !')
42         print('+-----+')
43         print('You already have an account. Directing to the login page.....')
44         login()
45     else:
46         customer_password = input("Password: ")
47         while len(customer_password) < 8 or len(customer_password) > 12:
48             print('\n+-----+')
49             print('!▲ Invalid password length. Please make sure it is between 8 to 12 digits! !')
50             print('+-----+\\n')
51             customer_password = input('Password: ')
52
53         while True:
54             try:
55                 age = int(input('Age: '))
56                 if age < 12:
57                     print('\n+-----+')
58                     print('!▲ You are under age. !')
59                     print('+-----+\\n')
60                 else:
61                     break
62             except ValueError:
63                 print('\n+-----+')
64                 print('!▲ Invalid age. Please enter numbers only. !')

```

Figure 4.4.1.2 Function of Sign Up

```

110 def login():
111     customer_name = input('\nName: ')
112
113     # Assume customer_info is loaded from customer.txt and contains details for each customer
114     if customer_name in customer_info:
115         # Check the account status of the customer
116         if customer_info[customer_name]['account_status'] == 'inactive':
117             print('\n+-----+')
118             print('!▲ Your account is INACTIVE. Please contact the manager. !')
119             print('+-----+\\n')
120             return None # Exit the login process if the account status is inactive
121
122         while True: # Loop until the correct username is entered
123             customer_username = input("Username: ")
124             # Check if the username matches the one in customer_info for the given customer_name
125             if customer_info[customer_name]['customer_username'] == customer_username:
126                 break # Exit the loop if the username is correct
127             else:
128                 print('\n+-----+')
129                 print('!▲ Incorrect username. Please enter again. !')
130                 print('+-----+\\n')
131
132             customer_password = input("Password: ")
133             while customer_password != customer_info[customer_name]['customer_password']:
134                 print('\n+-----+')
135                 print('!▲ Incorrect password. Please enter again. !')
136                 print('+-----+\\n')
137                 customer_password = input("Password: ")
138

```

Figure 4.4.1.3 Function of Log In

```

173     def update_personal_information():
174         if not customer_info:
175             print("No customer data available.")
176             return
177
178         while True: # Loop until the user enters the correct username
179             customer_username = input("Enter your username: ")
180
181             # Check if the username exists in customer_info
182             if customer_username in customer_info:
183                 customers = customer_info[customer_username] # Access the user's info dictionary
184                 print("What do you want to update?")
185                 print("1. Password")
186                 print("2. Age")
187                 print("3. Gender")
188                 print("4. Contact Number")
189                 print("5. Email")
190                 print("6. Address")
191
192                 choice = input("Choose the field number to update: ")
193
194                 if choice == "1":
195                     while True:
196                         new_password = input("Enter new password: ")
197                         if len(new_password) < 8 or len(new_password) > 12:
198                             print("\n+-----+")
199                             print('!▲ Invalid password length. Please make sure it is between 8 to 12 digits! !')
200                             print('+-----+\n')

```

Figure 4.4.1.4 Function of Update Personal Information

```

279     def account_management():
280         customer_info = load_data_from_customer()
281
282         if not customer_info: # Check if customer data is loaded
283             print("No customer data available.")
284             return
285
286         # Prompt for username input
287         customer_name = input("Please enter your name: ") # Convert to lowercase
288
289         # Check if the customer_name exists in the loaded customer data
290         if customer_name in customer_info:
291             print(f"Welcome, {customer_name}!")
292             print("What would you like to do?")
293             print("1. View Account Details")
294             print("2. Delete Account")
295             print("3. Exit to main menu")
296
297             choice = input("Enter your choice: ")
298
299             if choice == "1":
300                 # Display account details
301                 customer = customer_info[customer_name]
302                 print("\n+-----+")
303                 print(f"Username: {customer_name}")
304                 print(f"Password: {customer['customer_password']}") 
305                 print(f"Age: {customer['age']}") 
306                 print(f"Gender: {customer['gender']}") 
307                 print(f"Contact No: {customer['contact_no']}") 
308                 print(f"Email: {customer['email']}") 

```

Figure 4.4.1.5 Function of Account Management

In the customer account management function, customers can sign up for new accounts, log in, manage their accounts and update personal information. They can also perform other actions

such as browsing products, order tracking, viewing loyalty rewards, submitting product reviews or accessing the shopping cart based on their needs. Here, the customer menu serves as the main interface for customers to start their shopping process as it has provided 9 options for them to select and also some message will be print to guide them through the process as shown in figure 4.4.1.1. The ‘account_management()’ function allows customers to view their account details or delete their account. The ‘load_data_from_customer()’ function loads customer data from the customer.txt file, while the ‘save_info()’ function saves any updated information back to customer.txt. Also, the sign-up process includes validation checks to ensure that passwords and contact numbers follow a specific format while email, gender and age meet specified conditions and that customers cannot create duplicate accounts. In short, the customer menu function provides a smooth experience for customers by meeting their needs for account management and beginning their purchasing process.

Sample Output:

```
-----  
WELCOME TO MORNING GLORY BAKERY!  
-----  
1. Sign Up  
2. Login  
3. Browse Products  
4. Shopping Cart  
5. Order Tracking  
6. Submit Review  
7. View Loyalty Rewards  
8. Update Personal Information  
9. Manage Account  
0. Exit  
Please select an option (0-9): 2  
  
Name: Mark  
Username: Mark0825  
Password: 12345678  
  
Welcome back, Mark!
```

Figure 4.4.1.6 Sample Output of Log In

```
-----  
WELCOME TO MORNING GLORY BAKERY!  
-----  
1. Sign Up  
2. Login  
3. Browse Products  
4. Shopping Cart  
5. Order Tracking  
6. Submit Review  
7. View Loyalty Rewards  
8. Update Personal Information  
9. Manage Account  
0. Exit  
Please select an option (0-9): 1  
Name: Luna  
Username: Luna  
Password: 12345678  
Age: 20  
Gender (m=male, f=female, x=prefer not to say): f  
Contact number (xxx-xxxxxx): 012-1234567  
Email: luna04@gmail.com  
Address: 12, Maple street  
  
Information saved.  
Welcome, Luna! Your account has been created successfully!
```

Figure 4.4.1.7 Sample Output of Sign Up

```
Please select an option (0-9): 3  
  
-----  
PRODUCT BROWSING  
-----  
  
✿ Welcome to our bakery! We have a variety of product categories for you to explore:  
- Pastries  
- Biscuits  
- Breads  
- Cakes  
  
Feel free to browse the products by selecting a category or view the full menu!  
  
Please choose an option:  
1. Search for products by category  
2. View full menu  
3. Exit  
Enter your choice (1/2/3):
```

Figure 4.4.1.8 Sample Output of Selecting Product Browsing

```
4. Shopping Cart
5. Order Tracking
6. Submit Review
7. View Loyalty Rewards
8. Update Personal Information
9. Manage Account
0. Exit

Please select an option (0-9): 4
Hello, nct! Your cart ID is: 5977311891

-----
CART MANAGEMENT
-----

Please select an option:

1. Add item
2. Remove item
3. Modify item quantity in cart
4. View cart
5. Checkout or cancel
6. View payment receipt
7. Exit to main menu
Select your option:
```

Figure 4.4.1.9 Sample Output of Selecting Shopping Cart

```
1. Sign Up
2. Login
3. Browse Products
4. Shopping Cart
5. Order Tracking
6. Submit Review
7. View Loyalty Rewards
8. Update Personal Information
9. Manage Account
0. Exit

Please select an option (0-9): 5

-----
ORDER TRACKING
-----
Enter your Order ID: ORD001

Order Details
-----
Order ID:          ORD001
Username:         Mark0825
Items Ordered:    banana bread x 1, red velvet x 1
Total Price:      RM15.0
Status:           Payment Completed
-----
```

Figure 4.4.1.10 Sample Output of Selecting Order Tracking

```
Please select an option (0-9): 6

-----
          PRODUCT REVIEW
-----

Your recent purchases:
No.  Product Name           Quantity
-----
1    cheezy bun             10
2    cheezy bun             8
3    cheezy bun             3
4    gingerbread            2
5    cheezy bun             8
6    cheezy bun             6
7    cheezy bun             6
8    cheezy bun             5
9    gingerbread            5
10   gingerbread            2
11   cheezy bun             4
12   cheezy bun             5
13   gingerbread            5
14   cheezy bun             5
15   tiramisu               3

Select a product to review (enter the number): 1
Enter your review: good
Rate your product (1-5): |
```

Figure 4.4.1.11 Sample Output of Selecting Product Review

```
7. View Loyalty Rewards
8. Update Personal Information
9. Manage Account
0. Exit
Please select an option (0-9): 7

-----
          CUSTOMER LOYALTY REWARDS
-----

1. View Loyalty Rewards
2. Redeem Cash Vouchers
3. Exit to main menu
Select your option: 1
Enter your username: Mark0825

--Loyalty Rewards Information--

--Redeem History--

Order ID  | Total Spending (RM) | Points Earned | Status           | Redeem Rate (RM) | Vouchers Redeemed
-----
1        | 83.5                 | 4247          | MORNING GLORY'S GOLD | 1000          | 5
```

Figure 4.4.1.12 Sample Output of Selecting View Loyalty Rewards

```
-----  
WELCOME TO MORNING GLORY BAKERY!  
-----
```

1. Sign Up
2. Login
3. Browse Products
4. Shopping Cart
5. Order Tracking
6. Submit Review
7. View Loyalty Rewards
8. Update Personal Information
9. Manage Account

```
0. Exit
```

```
Please select an option (0-9): 8
```

```
Enter your username: nct
```

```
What do you want to update?
```

1. Password
2. Age
3. Gender
4. Contact Number
5. Email
6. Address

```
Choose the field number to update: 1
```

```
Enter new password: 123456789
```

```
Your information has been updated.
```

Figure 4.4.1.13 Sample Output of Selecting Update Personal Information

7. View Loyalty Rewards
8. Update Personal Information
9. Manage Account

```
0. Exit
```

```
Please select an option (0-9): 9
```

```
Please enter your name: nct
```

```
Welcome, nct!
```

```
What would you like to do?
```

1. View Account Details
2. Delete Account
3. Exit to main menu

```
Enter your choice: 1
```

```
+-----+
```

```
Username: nct
```

```
Password: 12345678
```

```
Age: 44
```

```
Gender: male
```

```
Contact No: 012-9999999
```

```
Email: s@gmail.com
```

```
Address: 2
```

```
Account status: active
```

```
Loyalty points: 5505
```

```
+-----+
```

Figure 4.4.1.14 Sample Output of Selecting Manage Account

In the customer account management function, customers can choose to sign up, manage their accounts, log in or update their personal information. The customer menu function allows customers to begin their journey by providing 9 options to select from. Customers can choose from the options displayed in the customer menu to sign up, log in, browse products, start shopping, track orders, submit reviews, view their loyalty rewards, update personal information or manage their accounts. Figure 4.4.1.6 shows the sample output of customer login, while Figure 4.4.1.7 shows the sample output of customer signup. Customers can select option 3 for product browsing and option 4 for the shopping cart, as shown in Figure 4.4.1.8 and Figure 4.4.1.9. Furthermore, based on Figure 4.4.1.10 and Figure 4.4.1.11, customers can select option 5 for order tracking and option 6 for submitting a product review. Also, customer can select option 7 to view loyalty rewards and option 8 to update their personal information based on figure 4.4.1.12 and figure 4.4.1.13. Based on figure 4.4.1.14, customers can select option 9 and enter their name to manage their account.

4.4.2 Product Browsing

```
40 def browse_products():
41     # Load data
42     categories = load_categories()
43     products = load_product_details()
44
45     print('\n-----')
46     print('\t\t\tPRODUCT BROWSING')
47     print('-----')
48
49     # Check if categories are available and display them to the user
50     if categories:
51         print("\n✿ Welcome to our bakery! We have a variety of product categories for you to explore:")
52         product_categories = set([item["category"] for item in categories.values()])
53         for category in product_categories:
54             print(f"- {category}")
55         print("\nFeel free to choose a category to browse the products or view the full menu!\n")
56     else:
57         print("No categories available at the moment.\n")
58
59     # Provide customers with choices
60     while True:
61         print()
62         print("Please choose an option:")
63         print("1. Search for products by category")
64         print("2. View full menu")
65         print("3. Exit")
66
67         choice = input("Enter your choice (1/2/3): ")
68
69         if choice == '1':
70             category_input = input("Please enter the product category you want to search for: ")
```

Figure 4.4.2.1 Product Browsing Function

The Product Browsing function allows customers to explore a variety of products available for purchase in our bakery. It starts by loading product categories from the baker_product_keeping.txt file and product details from the manager_product_inventory.txt file to provide customer with a list of available product categories and customers are given the options to search for products by category or view the full bakery menu. If a customer chooses to search by category, the system will check for available categories that match the customer's input and display product details such as product code, name, price and description using a while loop. Additionally, the system will validate the user's input and return to the menu if no matching categories are found. If the customer chooses to view the full menu, the complete list of products is displayed by importing product_menu.menu(). Finally, the system provides an option to exit if the customer finishes browsing. In short, this function can help customers easily browse all the bakery products.

Sample Output:

```
❖ Welcome to our bakery! We have a variety of product categories for you to explore:  
- Cakes  
- Breads  
- Biscuits  
- Pastries  
  
Feel free to choose a category to browse the products or view the full menu!  
  
Please choose an option:  
1. Search for products by category  
2. View full menu  
3. Exit  
Enter your choice (1/2/3): 1  
Please enter the product category you want to search for: Pastries  
  
Products in category 'Pastries':  
=====  
Product Code: 003  
Product Name: apple pie  
Price: RM 5.00  
Description: A mouthwatering dessert with tender filling covered by crispy crust  
-----
```

Figure 4.4.2.2 Sample Output of Select Products by Category

```
-----  
PRODUCT BROWSING  
-----  
  
❖ Welcome to our bakery! We have a variety of product categories for you to explore:  
- Cakes  
- Biscuits  
- Breads  
- Pastries  
  
Feel free to choose a category to browse the products or view the full menu!  
  
Please choose an option:  
1. Search for products by category  
2. View full menu  
3. Exit  
Enter your choice (1/2/3): 1  
Please enter the product category you want to search for: ice cream  
The category 'ice cream' does not exist.
```

Figure 4.4.2.3 Sample Output of Selecting Products by Category Not Shown

```

..... Morning Glory Bakery Menu ......

★ We offer a delightful selection of fresh breads, cakes, pastries, biscuits, and muffins, all baked daily to satisfy your cravings.
★ Explore our menu, and don't forget to check out our unique creations in the 'Others' category for something special!

↑ Breads ↑

BR890 - Banana Bread
Price: RM 4.50
Best Before: 17-08-2024
Allergen: R
👉 Rich in vitamin A and perfect for healthy treat

CH125 - Cheezy Bun
Price: RM 5.00
Best Before: 27-09-2024
Allergen: Cheese, Milk
👉 Extra gooey cheese for cheese aficionado

GR013 - Garlic Bun
Price: RM 5.60
Best Before: 06-11-2024
Allergen: Garlic
👉 A soft, fluffy bun infused with rich garlic butter for a savory, aromatic bite.

```

Figure 4.4.2.4 Sample Output of View Full Menu

Based on the figure 4.4.2.2, when the customer chooses option 1, they can select products by entering the category they wish to find and the results for the selected category will be displayed. If the category the customer is looking for does not exist, a message will be displayed indicating that the category was not found as shown in figure 4.4.2.3. Based on the figure 4.4.2.4, when customers select option 2, they can directly view the full menu of our bakery.

4.4.3 Cart Management

```
425     def shopping_cart(logged_in_username): # Accept the logged-in username as an argument
426         cart = {} # Initialize cart as a regular dictionary
427         cart_id = generate_cart_id() # Generate a 10-digit numeric cart ID
428         print(f"Hello, {logged_in_username}! Your cart ID is: {cart_id}\n") # Use logged_in_username
429
430         print('\n-----')
431         print('|\t|\t|', ' ', 'CART MANAGEMENT')
432         print('-----')
433
434         while True:
435             print("\n Please select an option:")
436             print()
437             print("1. Add item")
438             print("2. Remove item")
439             print("3. Modify item quantity in cart")
440             print("4. View cart")
441             print("5. Checkout or cancel")
442             print("6. View payment receipt")
443             print("7. Exit to main menu")
444
445             option = input("Select your option: ").strip()
446
447             if option == '1':
448                 add_item_to_cart(cart)
449             elif option == '2':
450                 remove_item_from_cart(cart)
451             elif option == '3':
452                 modify_item_quantity(cart)
453             elif option == '4':
454                 view_cart(cart)
```

Figure 4.4.3.1 Function of Cart Management

```
78     def add_item_to_cart(cart):
79         while True: # Loop to allow re-entering product code if the user chooses 'no'
80             product_menu.menu()
81             baker_data = load_baker_data() # Load baker product data
82             manager_data = load_manager_data() # Load manager product data
83             discount_data = load_discount_data() # Load discount data
84
85             product_code_input = input("\nEnter the product code: ").strip()
86
87             # Directly search for the product in baker_data
88             for item in baker_data.values():
89                 if item['product_code'] == product_code_input:
90                     product_name = item["product_name"] # Retrieve product name
91
92                     # Get the price from manager_data
93                     for key, value in manager_data.items():
94                         if value['product_name'] == product_name:
95                             product_price = float(value.get("price", 0).replace('RM ', '')) # Convert price to float
96                             break # Exit loop once the product is found
97
98                     # Check if the product has a discount in discount_data
99                     discount_percentage = 0 # Default to 0% discount if no discount found
100                    for discount_item in discount_data.values():
101                        if discount_item['product_code'] == product_code_input:
102                            discount_percentage = float(discount_item['Discount'].replace('%', '')) / 100
103                            break
```

Figure 4.4.3.2 Function of Add Item to Cart

```

165 def remove_item_from_cart(cart):
166     # Check if the cart is empty
167     if not cart:
168         print("⚠ Your cart is empty. There are no items to remove.")
169         return # Exit the function if the cart is empty
170
171     while True:
172         product_code = input("\nPlease enter the product code of the item you wish to remove: ").strip()
173
174         # Check if the product exists in the cart
175         if product_code in cart:
176             del cart[product_code] # Remove the item from the cart
177             print(f"Product {product_code} has been removed from the cart successfully!")
178         else:
179             print("⚠ Product cannot be found in the cart.")
180
181         # Ask if the user wants to continue removing items
182         continue_removing = input(
183             "\nDo you want to continue removing from the cart? (yes = y / no = n): ").lower().strip()
184
185         # Check user input for continuation or exit
186         if continue_removing in ['y', 'yes']:
187             continue # Continue the loop to remove more items
188         elif continue_removing in ['n', 'no']:
189             print("Exiting item removal process.")
190             break # Exit the removal process
191         else:
192             print("⚠ Invalid input! Please enter 'y' for yes or 'n' for no.|")

```

Figure 4.4.3.3 Function of Remove Item from Cart

```

194 def modify_item_quantity(cart):
195     if not cart: # Check if the cart is empty
196         print("⚠ Your cart is empty. You cannot modify item quantities.|")
197         return # Exit the function if the cart is empty
198
199     while True:
200         product_code = input("\nPlease enter the product code of the item you wish to modify: ").strip()
201
202         if product_code in cart:
203             while True:
204                 try:
205                     new_quantity = int(input(f"Enter new quantity for {cart[product_code]['product_name']}: "))
206                     if new_quantity < 0:
207                         print("Quantity cannot be negative.")
208                         continue # Prompt again for a valid quantity
209                     # Update quantity
210                     cart[product_code]['quantity'] = new_quantity
211
212                     # Calculate new total price
213                     new_total_price = new_quantity * cart[product_code]['price']
214
215                     # Display updated information and new total price
216                     print(f"Updated {cart[product_code]['product_name']} quantity to {new_quantity}.")
217                     print(f"New total price for {cart[product_code]['product_name']}: RM {new_total_price:.2f}")
218                     break # Exit the quantity loop after a successful update
219
220                 except ValueError:
221                     print("⚠ Invalid input! Please enter a valid number.|")

```

Figure 4.4.3.4 Function of Modify Item Quantity

```

239 def view_cart(cart):
240     while True: # Loop to allow re-viewing the cart if 'no' is selected
241         if not cart:
242             print("\nYour cart is empty.") # Print this message if the cart is empty
243             return
244
245         print("\nYour Cart:")
246         print()
247         print(f"{'Product':<20} | {'Quantity':<8} | Price (RM)")
248         print("-" * 40)
249
250         for item in cart.values():
251             item_total = item['quantity'] * item['price']
252             print(f"{item['product_name']:<20} | {item['quantity']:<8} | RM{item_total:.2f}")
253
254         print("-" * 40)
255
256         # Prompt the user if they want to return to the main menu
257         choice = input("\nDo you want to return to the main menu? (y/n): ").lower()
258
259         if choice == 'y':
260             print("Returning to the main menu...")
261             return # Exit the function and return to the main menu
262         elif choice == 'n':
263             print("Here is your cart again.\n") # Reloop to show the cart again
264         else:
265             print("\n+-----+")
266             print("|\u25bc Invalid input. Please enter 'y' or 'n'. |")
267             print("+-----+\n")

```

Figure 4.4.3.5 Function of View Cart

```

307 def checkout_or_cancel(cart, customer_name, cart_id):
308     if not cart:
309         print("\nYour cart is empty! Please add items before proceeding to checkout.")
310         return
311
312     total_price = display.cart(cart) # Displays and calculates total price
313     print(f"Current cart ID: {cart_id}")
314
315     print("\nWould you like to:")
316     print("1. Place order")
317     print("2. Cancel your order")
318
319     choice = input("Please select your option (1 or 2): ").strip()
320     print(f"User choice: {choice}") # Debugging choice
321
322     # Load existing order data
323     try:
324         with open("customer_order_list.txt", "r") as file:
325             order_data = json.load(file)
326     except FileNotFoundError:
327         order_data = []
328
329     order_id = generate_order_id(order_data) # Generate order ID
330
331     if choice == '1':
332         print(f"\nTotal price to pay: RM {total_price:.2f}")
333         print("\nOrder placed. Please proceed to payment to view the receipt.")
334

```

Figure 4.4.3.6 Function of Checkout or Cancel

```

1 usage  ▲ Ng Yvonne
414 def view_payment_receipt(cart_id):
415     with open('customer_order_list.txt', 'r') as file:
416         order_list = json.load(file)
417         order = order_list.get(str(cart_id))
418         if order and order['status'] == 'Payment Completed':
419             cashier_transaction_completion.receipt(str(cart_id))
420         else:
421             print('\nThe receipt will only be generated after payment has completed. Please proceed to payment at the counter or via online.')

```

Figure 4.4.3.7 Function of View Payment Receipt

The cart management function aims to enable customers to manage their shopping cart such as add, remove, view or modify items in their shopping cart, proceed to checkout or cancel and view payment receipt efficiently. In the ‘shopping_cart(logged_in_username)’ function, customers can choose their desired actions by selecting from the given options. By presenting these options clearly, customers can easily navigate the cart management process and make informed decisions about their orders. The ‘add_item_to_cart (cart)’ function and the ‘remove_item_from_cart (cart)’ function provide the ability to add or remove items ensures that customers can adjust their selections anytime. Also, the ‘modify_item_quantity()’ function allows customers to efficiently modify the item quantity in the cart if they change their mind. Furthermore, the ‘view_cart()’ function displays all the items in the customer’s cart to help them confirm and review items in their cart before proceeding to checkout. Furthermore, the ‘checkout_or_cancel()’ function enables customers to choose whether to proceed to payment or cancel their order, while the ‘view_payment_receipt()’ function allows customers to view their payment receipt after their status becomes ‘Payment Completed’.

Sample Output:

```
Enter the product code: CK858
How many tiramisu would you like to add? 3

tiramisu x3 has been added to your cart.
Total price: RM 8.5 x 3 = RM 25.5

Do you want to return to the main menu? (y/n): Y
Returning to the main menu...

Please select an option:

1. Add item
2. Remove item
3. Modify item quantity in cart
4. View cart
5. Checkout or cancel
6. View payment receipt
7. Exit to main menu
Select your option:
```

Figure 4.4.3.8 Sample Output of Add Items to Cart

```
Please select an option:

1. Add item
2. Remove item
3. Modify item quantity in cart
4. View cart
5. Checkout or cancel
6. View payment receipt
7. Exit to main menu
Select your option: 2

Please enter the product code of the item you wish to remove: CK858
Product CK858 has been removed from the cart successfully!

Do you want to continue removing from the cart? (yes = y / no = n):
```

Figure 4.4.3.9 Sample Output of Remove Items to Cart

```
Please select an option:  
  
1. Add item  
2. Remove item  
3. Modify item quantity in cart  
4. View cart  
5. Checkout or cancel  
6. View payment receipt  
7. Exit to main menu  
Select your option: 3  
  
Please enter the product code of the item you wish to modify: BS678  
Enter new quantity for gingerbread: 2  
Updated gingerbread quantity to 2.  
New total price for gingerbread: RM 4.50  
  
Do you want to continue modifying item quantities? (yes = y / no = n): N  
Exiting item quantity modification process.
```

Figure 4.4.3.10 Sample Output of Modify Item Quantity in Cart

```
Please select an option:  
  
1. Add item  
2. Remove item  
3. Modify item quantity in cart  
4. View cart  
5. Checkout or cancel  
6. View payment receipt  
7. Exit to main menu  
Select your option: 4  
  
Your Cart:  
  
Product | Quantity | Price (RM)  
-----  
gingerbread | 2 | RM4.50  
tiramisu | 2 | RM17.00  
-----  
  
Do you want to return to the main menu? (y/n): |
```

Figure 4.4.3.11 Sample Output of View Cart

```
3. Modify item quantity in cart
4. View cart
5. Checkout or cancel
6. View payment receipt
7. Exit to main menu
Select your option: 5

Your cart contains:
tiramisu x 4 at RM 8.50

Total price: RM 34.00
Current cart ID: 4664798155

Would you like to:
1. Place order
2. Cancel your order
Please select your option (1 or 2): 1
User choice: 1

Total price to pay: RM 34.00

Order placed. The receipt will be generated after payment is completed. Please pay at the counter or online.
Order has been saved successfully.
net's loyalty points updated to: 5845
Customer loyalty points updated in customer.txt.
Order placed. Please proceed to payment for the receipt.
```

Figure 4.4.3.12 Sample Output of Checkout

```
1. Add item
2. Remove item
3. Modify item quantity in cart
4. View cart
5. Checkout or cancel
6. View payment receipt
7. Exit to main menu
Select your option: 5

Your cart contains:
tiramisu x 3 at RM 8.50

Total price: RM 25.50
Current cart ID: 6714479214

Would you like to:
1. Place order
2. Cancel your order
Please select your option (1 or 2): 2
User choice: 2
Order has been successfully canceled!
```

Figure 4.4.3.13 Sample Output of Cancel Order

MORNING GLORY BAKERY
Reg: 219875000123 (80851-Z)
51, Lorong Maplewood, Taman Springfield, 47180, Puchong, Selangor.
Tel: 04-5678951
Email: morningglorybakery@gmail.com
Business hours: 9.00AM - 6.00PM

03:01:28 PM

26-10-2024

Receipt no : MGB-5481360
Username : chew
Order ID : ORD044
Cart ID : 4992374927

Item	Qty	Price(RM)	Amount(RM)
Cheezy Bun	2	5.00	10.00
Gingerbread	2	2.50	5.00
		Subtotal:	15.00
		Discounted price:	13.50
		Service tax @ 6%:	0.81
Points earned: 135		TOTAL:	14.31

Goods sold are not returnable and refundable !
***** Thank You Please Come Again *****

Figure 4.4.3.14 Sample Output of View Payment Receipt

In the cart management function, customers can choose their options from the shopping cart menu. By selecting option 1, users can add items they want to the cart as shown in figure 4.4.3.8, and option 2 allows them to remove items if they change their mind as shown in figure 4.4.3.9. Additionally, users can modify the item quantity in their cart with option 3 and view the items in their cart with option 4 as shown in figure 4.4.3.10 and figure 4.4.3.11. To complete the checkout or cancel their order, users should choose option 5 to proceed with payment or cancel the order as shown in figure 4.4.3.12 and figure 4.4.3.13. Finally, after completing the payment, customers can view their payment receipt by selecting option 6 as shown in figure 4.4.3.14.

4.4.4 Order Tracking

```
1 usage ± XH*
2 def order_tracking():
3     orders = load_data_from_tracking()
4     print('\n-----')
5     print('\t\t\t', 'ORDER TRACKING')
6     print('-----')
7
8     while True:
9         order_id = input('Enter your Order ID: ') # Prompt the customer to enter their order ID
10
11         # Search for the order by order_id
12         order_found = False
13         for order_key, order in orders.items():
14             if order['order_id'] == order_id: # Compare the input with the order_id in the order details
15                 print(f'\n{"Order Details":^30}')
16                 print('-' * 55)
17                 print(f'{"Order ID":<20} {order_id}')
18                 print(f'{"Username":<20} {order["username"]}')
19                 print(f'{"Items Ordered":<20} {" ".join(order["items_ordered"])}')
20                 print(f'{"Total Price":<20} RM{order["total_price (RM)":.1f]}')
21                 print(f'{"Status":<20} {order["status"]}')
22                 print('-' * 55)
23                 order_found = True
24                 break # Exit the loop when a matching order is found
25
26             if order_found:
27                 break # Exit the loop after displaying the order details
28
29         # Display this message if Order ID cannot be found
30         print('|\u25bc Order ID cannot be found. Please check and try again!|')
```

Figure 4.4.4.1 Order Tracking Function

The order tracking function aims to enable customers to track the status of their orders through the system. First, the ‘load_data_from_tracking()’ function loads the order data and retrieves all order records stored in the system. After loading the data, customer need to enter their Order ID to searches for their order in the dictionary. This ID serves as a unique identifier for the order because if the order is found, the order details information such as the order ID, username, items ordered, total price and current status of the order such as Order Placed, Payment Completed or Cancelled will display. If the order ID does not exist, an error message is displayed to inform the user that the Order ID cannot be found to allow customer to double-check their entry and try again. Overall, this function ensure that customers can check their order status anytime and can clearly view their order details such as the order status by entering their Order ID.

Sample Output:

```
-----  
          ORDER TRACKING  
-----  
Enter your Order ID: ORD001  
  
      Order Details  
-----  
Order ID:          ORD001  
Username:         Mark0825  
Items Ordered:    banana bread x 1, red velvet x 1  
Total Price:      RM15.0  
Status:           Order Placed  
-----
```

Figure 4.4.4.2 Sample Output of Order Tracking

```
-----  
          ORDER TRACKING  
-----  
Enter your Order ID: ORD abc  
|⚠ Order ID cannot be found. Please check and try again!|  
Enter your Order ID: |
```

Figure 4.4.4.3 Sample Output of Order Tracking Not Shown

As shown in figure 4.4.4.2, when customers enter a correct order ID, they can view their order details such as items ordered, total price and status. If the order ID entered by customers is incorrect or cannot be found, an error message will be shown, and the user will need to enter their order ID again as shown in figure 4.4.4.3.

4.4.5 Product Review

```
44 def submit_review(logged_in_username):
45     print('\n-----')
46     print(' \t\t\tPRODUCT REVIEW')
47     print('-----')
48
49     order_list = load_order_list()
50     recent_purchases = []
51     order_date = None
52
53     # Check the user's purchase history
54     for order_data in order_list.values():
55         if order_data["username"] == logged_in_username and order_data["status"] == "Order Placed":
56             recent_purchases.extend(order_data['items_ordered'])
57             order_date = order_data['order_date']
58
59     if not recent_purchases:
60         print('|\u25bc You have not completed any purchases. Please buy something before writing a review!|')
61         return
62
63     # Display recent purchases
64     print("Your recent purchases:")
65     print(f"\u00b7{'No.':<5} {'Product Name':<30} {'Quantity':<10}")
66     print('-' * 60)
67
68     for i, product in enumerate(recent_purchases, start=1):
69         product_name, quantity = product.split(' x ')
70         print(f"\u00b7{i:<5} {product_name:<30} {quantity:<10}")
```

Figure 4.4.5.1 Product Review Function

The Customer Product Review function allows customers to provide feedback and suggestions about products they have purchase. First, the ‘load_review()’ function will load existing reviews by reading them from the customer_reviews.txt file. Besides, the ‘submit_review(logged_in_username)’ function checks if the customer has a purchase record by loading their order history from the customer_order_list.txt file using the ‘load_order_list()’ function, and also ensures that only logged-in customers with a purchase history can submit a review. The customer can write a review and rate the product only if a purchase is found with the status “Payment Completed.”. If no record is found, the function will print a message informing the customer that they haven’t made any purchases yet and need to buy something first before submitting a review. Also, the ‘validate_rating (rating)’ function ensures that the rating given by the customer is a valid number between 1 and 5. The review and rating are then stored in the reviews dictionary. Customers who do not have any purchases will not be able to submit a review. Finally, the updated review is saved back to the customer_reviews.txt file without replacing the previous comments. The product review feature provides an organized way to store reviews and allows customers to easily submit reviews after completing a purchase.

Sample Output:

```
-----  
WELCOME TO MORNING GLORY BAKERY!  
-----  
1. Sign Up  
2. Login  
3. Browse Products  
4. Shopping Cart  
5. Order Tracking  
6. Submit Review  
7. View Loyalty Rewards  
8. Update Personal Information  
9. Manage Account  
0. Exit  
Please select an option (0-9): 6  
You need to log in first to submit a review.
```

Figure 4.5.5.2 Sample Output of Product Review Not Shown Without Logging In

```
-----  
WELCOME TO MORNING GLORY BAKERY!  
-----  
1. Sign Up  
2. Login  
3. Browse Products  
4. Shopping Cart  
5. Order Tracking  
6. Submit Review  
7. View Loyalty Rewards  
8. Update Personal Information  
9. Manage Account  
0. Exit  
Please select an option (0-9): 6  
  
-----  
PRODUCT REVIEW  
-----  
|⚠ You have not completed any purchases. Please buy something before writing a review!|
```

Figure 4.5.5.3 Sample Output of Product Review Without Completing Any Purchases

```

-----  

      PRODUCT REVIEW  

-----  

Your recent purchases:  

No.   Product Name           Quantity  

-----  

1     banana bread           1  

2     red velvet              1  

3     cheezy bun              6  

4     cheezy bun              6  

5     cheezy bun              6  

Select a product to review (enter the number): 2  

Enter your review: good  

Rate your product (1-5): 5  

***** Thank you for your feedback! *****  

Your review has been successfully received.

```

Figure 4.5.5.4 Sample Output of Product Review

In the product review function, when customers select option 6 from the customer menu to submit a review without logging into their account, they will be unable to submit a review and a reminder to log in will be displayed as shown in figure 4.5.5.2. Based on figure 4.5.5.3, a new customer who has signed up but haven't completed any purchases will also be unable to submit a review. Only customers with a purchase history and who are logged in will be able to submit a review as shown in figure 4.5.5.4.

4.4.6 Customer Loyalty Rewards

```
260     def loyalty_rewards():
261         while True:
262             print('\n-----')
263             print(' \t\t\tCUSTOMER LOYALTY REWARDS')
264             print('-----')
265             print()
266             print("1. View Loyalty Rewards")
267             print("2. Redeem Cash Vouchers")
268             print("3. Exit to main menu")
269
270             choice = input("Select your option: ")
271
272             if choice == '1':
273                 username = input("Enter your username: ").strip()
274                 view_loyalty_rewards(username)
275
276             elif choice == '2':
277                 username = input("Enter your username: ").strip()
278                 redeem_cash_voucher(username)
279
280             elif choice == '3':
281                 print("Thank you for visiting the Customer Loyalty Program. Goodbye!")
282                 break
283
284             else:
285                 print("|⚠ Invalid option! Please try again.|")
```

Figure 4.4.6.1 Function of Customer Loyalty Rewards

```
85     def process_payment(username, total_price): # Process payment and update the user's loyalty points and status
86         rewards = load_loyalty_rewards()
87         customers = load_customer_data()
88
89         # Ensure total_price is a valid number
90         try:
91             total_price = float(total_price)
92         except ValueError:
93             print(f"\t⚠ Error: total_price must be a valid number, got {total_price}!")
94             return
95
96         # Check if user exists in customer_loyalty_rewards.txt
97         user_rewards = [order_id for order_id, data in rewards.items() if data['username'] == username]
98
99         # Determine new loyalty points
100        points_change = determine_loyalty_points(total_price)
101
102        if not user_rewards:
103            # New user, initialize their data in customer_loyalty_rewards.txt
104            new_order_id = str(len(rewards) + 1) # Generate a new order ID
105            rewards[new_order_id] = {
106                "username": username,
107                "total_spending (RM)": total_price,
108                "loyalty_points": points_change,
109                "status": "MORNING GLORY'S STANDARD",
110                "redeem_rate (RM)": REDEEM_RATES["MORNING GLORY'S STANDARD"], # Set initial redeem rate
111                "voucher_redeem": 0,
112                "redeem_history": []
113            }
```

Figure 4.4.6.2 Function of Process Payment

```

182 def redeem_cash_voucher(username): # Allow customers to redeem cash vouchers using loyalty points
183     rewards = load_loyalty_rewards()
184
185     for order_id, history in rewards.items():
186         if history['username'] == username:
187             points = history['loyalty_points']
188             status = history['status']
189             points_per_voucher = REDEEM_RATES.get(status)
190
191             if points_per_voucher is None:
192                 print("Standard users cannot redeem cash vouchers.")
193                 return
194
195             max_vouchers = points // points_per_voucher # Each cash voucher is worth RM10 in the loyalty program
196             if max_vouchers > 0:
197                 print(f"You have {points} loyalty points.")
198                 print(f"As a {status} member, you can redeem up to {max_vouchers} vouchers (Each RM10).")
199                 choice = input("Would you like to redeem vouchers? (yes=y, no=n): ").lower()
200
201             if choice == 'y':
202                 try:
203                     num_vouchers = int(input(f"How many vouchers? (up to {max_vouchers}): "))
204                     if 0 < num_vouchers <= max_vouchers:
205                         total_points_deducted = num_vouchers * points_per_voucher
206                         history['loyalty_points'] -= total_points_deducted
207                         print(f"Success! Redeemed {num_vouchers} voucher(s) worth RM{num_vouchers * 10}.")
208
209                         if 'voucher_redeem' not in history:
210
211

```

Figure 4.4.6.3 Function of Redeem Cash Voucher

```

258 def view_loyalty_rewards(username): #Display loyalty rewards information for a specific user
259     rewards = load_loyalty_rewards() # Load from customer_loyalty_rewards.txt
260
261     # Check if the username exists in the rewards data
262     user_rewards = [reward for order_id, reward in rewards.items() if reward['username'] == username]
263
264     if user_rewards:
265         print('\n--Loyalty Rewards Information-- ')
266         print()
267         print("\n--Redeem History--")
268         print('-' * 150)
269         header = f"{'Order ID':<10} | {'Total Spending (RM)':<20} | {'Points Earned':<15} | {'Status':<20} | {'Redeem Rate (RM)':<15} | {'Vouchers Redeemed':<15}"
270         print(header)
271         print('-' * 150)
272         # Display the values in aligned format
273         for order_id, r in rewards.items():
274             if r['username'] == username:
275                 points_earned = r.get('loyalty_points', '-')
276                 voucher_redeemed = r.get('voucher_redeem', '-')
277                 status = r.get('status', '-') # Get status to avoid KeyError
278                 redeem_rate = r.get('redeem_rate (RM)', '-') # Get redeem rate to avoid KeyError
279                 print(
280                     f'{order_id:<10} | {r["total_spending (RM)":<20} | {points_earned:<15} | {status:<20} | {redeem_rate:<15} | {voucher_redeemed:<15}')
281
282         # Display redemption history
283     else:
284         print(f"No loyalty rewards found for username: {username}")

```

Figure 4.4.6.4 Function of View Loyalty Rewards

```

# XH *
144 def update_customer_loyalty_points(customer_name, points_change): # Update loyalty points for customer
145     try:
146         with open("customer.txt", "r") as file:
147             customer_data = json.load(file)
148
149             for customer_id, customer in customer_data.items():
150                 if customer['customer_username'] == customer_name:
151                     customer['loyalty_points'] += points_change
152                     new_status = update_customer_status(customer['loyalty_points'])
153                     break
154
155             save_customer_data(customer_data)
156             update_loyalty_rewards(customer_name, points_change, new_status)
157
158     except (FileNotFoundException, json.JSONDecodeError) as e:
159         print(f"Error: {e}")

```

Figure 4.4.6.5 Function of Update Customer Loyalty Points

```

usage: ±XH
166 def update_loyalty_rewards(username, points_change, new_status): # Update loyalty rewards for a user
167     rewards = load_loyalty_rewards()
168
169     for order_id, history in rewards.items():
170         if history['username'] == username:
171             history['loyalty_points'] += points_change
172             history['status'] = new_status
173             # Keep the redeem rate fixed as initially set, regardless of status upgrade
174             if 'redeem_rate (RM)' not in history or history['redeem_rate (RM)'] == 0:
175                 history['redeem_rate (RM)'] = REDEEM_RATES.get(new_status, 0)
176             break
177
178     save_loyalty_rewards(rewards)
179     print("Loyalty rewards updated.")

```

Figure 4.4.6.6 Function of Update Loyalty Rewards

The customer loyalty rewards function manages various tasks such as calculating customer loyalty points, determining and updating customer loyalty status, and saving these details in the customer_loyalty_rewards.txt file to allow customers to enjoy loyalty rewards. Through the loyalty program, customers earn points based on their total spending and can redeem cash vouchers with these points. For example, customers can earn 10 points for every RM spent as they progress through four status tiers: Standard, Bronze at 500 points, Silver at 1000 points, and Gold at 2000 points. Each tier offers specific redemption rates: Gold customers can redeem RM10 for every 800 points, Silver customers for every 900 points, and Bronze customers for every 1000 points. Standard customers do not earn redeemable points. The ‘process_payment(username, total_price)’ function calculates loyalty points based on the customer's total spending, manages customer status and sets cash voucher redemption rates

according to loyalty status. The ‘update_customer_status()’ function automatically adjusts the customer's status to Standard, Bronze, Silver or Gold based on their accumulated points. The ‘update_customer_loyalty_points(customer_name, points_change)’ function will updates loyalty points in the customer.txt file for customer while the ‘update_loyalty_rewards(username, points_change, new_status)’ function will updates loyalty rewards data in customer_loyalty_rewards.txt. Besides, the ‘redeem_cash_voucher()’ function allows customers to redeem vouchers using loyalty points according to their status. When a voucher is redeemed, the appropriate points are deducted, and the redemption is recorded in the customer's redeem history. Additionally, the ‘view_loyalty_rewards()’ function enables customers to check their loyalty information such as total spending, loyalty points, redemption rate, status and the number of vouchers redeemed. With the customer loyalty program, customer engagement can increase, and customers can easily view their redemption history or redeem cash vouchers at any time.

Sample Output:

```
CUSTOMER LOYALTY REWARDS
-----
1. View Loyalty Rewards
2. Redeem Cash Vouchers
3. Exit to main menu
Select your option: 1
Enter your username: Mark0825
--Loyalty Rewards Information--
--Redeem History--
-----
Order ID | Total Spending (RM) | Points Earned | Status | Redeem Rate (RM) | Vouchers Redeemed
-----
1 | 83.5 | 4547 | MORNING GLORY'S GOLD | 120 | 2
```

Figure 4.4.6.7 Sample Output of View Loyalty Rewards

```
CUSTOMER LOYALTY REWARDS

1. View Loyalty Rewards
2. Redeem Cash Vouchers
3. Exit to main menu
Select your option: 2
Enter your username: nct
You have 6695 loyalty points.
As a MORNING GLORY'S GOLD member, you can redeem up to 8 vouchers (Each RM10).
Would you like to redeem vouchers? (yes=y, no=n): y
How many vouchers? (up to 8): 2
Success! Redeemed 2 voucher(s) worth RM20.
```

Figure 4.4.6.8 Sample Output of Redeem Cash Vouchers

```
CUSTOMER LOYALTY REWARDS

1. View Loyalty Rewards
2. Redeem Cash Vouchers
3. Exit to main menu
Select your option: 2
Enter your username: Mark0825
You have 4247 loyalty points.
As a MORNING GLORY'S GOLD member, you can redeem up to 42 vouchers (Each RM10).
Would you like to redeem vouchers? (yes=y, no=n): n
No vouchers redeemed.
```

Figure 4.4.6.9 Sample Output of No Redemption for Cash Vouchers

```
CUSTOMER LOYALTY REWARDS

1. View Loyalty Rewards
2. Redeem Cash Vouchers
3. Exit to main menu
Select your option: 2
Enter your username: Mark0825
You have 4247 loyalty points.
As a MORNING GLORY'S GOLD member, you can redeem up to 42 vouchers (Each RM10).
Would you like to redeem vouchers? (yes=y, no=n): y
How many vouchers? (up to 42): 44
Invalid number of vouchers.
```

Figure 4.4.6.10 Sample Output of Redeem Invalid number of Cash Vouchers

In the customer loyalty rewards system, customers can view their rewards with details on total spending, points earned, status, redeem rate and the number of vouchers redeemed as shown in

figure 4.4.6.7. If the username is invalid, they will not be able to access their rewards. Based on Figure 4.4.6.8 and figure 4.4.6.9, customers can also choose whether to redeem cash vouchers and enter the number of vouchers they want to redeem if they wish to do so. If the number they input to redeem is invalid, an error message will be displayed, and they will be unable to redeem as shown in Figure 4.4.6.10.

4.5 Cashier

4.5.1 Sign In/ Log In

```
43 # cashier account login function
44 usages = Ng Yvonne +1
45
46 def cashier_accounts():
47
48     print('')
49     printed_centered('CASHIER')
50     cashier_username = input('Username: ') # ask for username
51     if cashier_username not in cashier_info: # continue looping if username not match with the name
52         print('\n+-----+')
53         print('|\u25bc You are not a cashier, cannot access to cashier privilege. |')
54         print('+-----+\n')
55         print('Exiting to Main Page.....')
56         return False
57
58     else:
59         cashier_password = input('Password: ') # ask for cashier's password
60         while len(cashier_password) < 8 or len(cashier_password) > 12: # repeating the prompt to input when password length is invalid
61             print('\n+-----+')
62             print('|\u25bc Invalid password length. Please make sure it is between 8 to 12 digits! |')
63             print('+-----+\n')
64             cashier_password = input('Password: ')
65
66         while cashier_password != cashier_info[cashier_username]['cashier_password']: # continue to validate the password
67             print('\n+-----+')
68             print('|\u25bc Incorrect password. Please enter again. |')
69             print('+-----+\n')
70             cashier_password = input('Password: ')
71
72     print('\nLogin successfully!') # login successfully if the password meet the 2 requirements above
73     print(f'Welcome, cashier {cashier_username}!')
74     # display cashier privilege
75     while True:
```

Figure 4.5.1.1 Cashier login or sign in function

The function begins by prompting user to enter cashier username. Next, it will validate user input by matching it with recorded cashier username in cashier txt file. If the function unable to match the user input with cashier username stored in cashier file, it will exit the loop and return to the main page. If matches, it will then prompt user to enter password and match the input again with the password of user selected cashier. If the password matches, it will display the cashier privilege menu with options for product displaying, managing discount, completing transaction, generating sales report, and return to main page. Corresponding function is called based on user input. If user enter an invalid input, it will display error message and prompt user again.

Sample Output:

```
-----  
          CASHIER  
-----  
  
Username: minion  
  
+-----+  
|⚠ You are not a cashier, cannot access to cashier privilege. |  
+-----+  
  
Exiting to Main Page.....  
  
Role Options:  
1. Manager 🏢👤  
2. Baker 🧑🍞  
3. Cashier 💵🟡  
4. Customer 🤴🟡  
5. Exit the program ➡ BACK  
  
Who are you? (1, 2, 3, 4):  
>>> |
```

Figure 4.5.1.2 Sample output of invalid cashier username

```
-----  
          CASHIER  
-----  
  
Username: verticca  
Password: securec@sh12  
  
Login successfully!  
Welcome, cashier verticca!  
  
-----  
          CASHIER PRIVILEGE  
-----  
a. Product display  
b. Manage discount  
c. Transaction completion  
d. Reporting  
e. Back to Main page  
  
Select a choice (a, b, c, d, e):  
>>> |
```

Figure 4.5.1.3 Sample output of valid cashier username and password

Figure 4.5.1.2 shows the scenario when the function cannot match username input by user with any recorded cashier data. Figure 4.6.5.3 shows the scenario when user enter a valid cashier

username and password, where the function will display the cashier privilege menu and call the function based on user input options.

4.5.2 Product Display

```
80     # define the function to display menu
81     3 usages: ▲ hxdum +l
82     def menu():
83         print('\n' + '*' * 55, ' Morning Glory Bakery Menu ', '*' * 55, '\n') # title of menu
84         print(
85             '★ We offer a delightful selection of fresh breads, cakes, pastries, biscuits, and muffins, all baked daily to satisfy your cravings.')
86         print(
87             '★ Explore our menu, and don't forget to check out our unique creations in the "Others" category for something special!\n')
88
89     # group the product by their category for organized display
90     category_groups = {}
91     product_details = 'manager_product_inventory.txt'
92
93     for value in product_data.values():
94         category = value['category']
95
96         # if the category does not exist in the dictionary, create a new list for that category
97         if category not in category_groups:
98             category_groups[category] = []
99
100        # if category exist, append the product after format data
101        category_groups[category].append(format_product_data(value, product_details).split('\n'))
102
103    # loop through each category to print its product
104    for category, products in category_groups.items():
105
106        print(f'\n\t {category} \t\n')
107
108        # display the product in pairs of two
109        for i in range(0, len(products), 2):
110            product1 = wrap_data(products[i], width=65)
111            product2 = wrap_data(products[i+1], width=65)
112            print(f'{product1}\t{product2}'
```

Figure 4.5.2.1 Function that display product menu

The function starts by printing a title and brief description about the bakery offerings. Next, it gets product data from the baker_product_keeping file and group them by their category. An empty dictionary, category_groups, is initialized to hold this data. The function loop through each product from baker_product_keeping file and extracts its category. If the category does not exist in category_groups dictionary, it creates a new list for that category. Each product data is then passed to format_product_details() function for formatting.

The format_product_details() function get the product data and product inventory data by reading from manager_product_inventory file, and format them to display the product name, price, expiry date, allergens and description. Once formatted, the product data is added to its respective category list in category_groups dictionary.

To improve readability, products are displayed two at a time, side-by-side in columns. Using wrap_data() function with a width of 65 characters, each product data is wrapped for consistency in presentation. If there is an odd number of products, the last product is displayed

on its own. The `print_in_column()` function handles the side-by-side printing and ensure each product is printed in pairs. Once all product in a category have been displayed, a separator line is printed to provide a clear visual break between categories.

Sample Output:

```
..... Morning Glory Bakery Menu .....
```

```
❖ We offer a delightful selection of fresh breads, cakes, pastries, biscuits, and muffins, all baked daily to satisfy your cravings.  
❖ Explore our menu, and don't forget to check out our unique creations in the 'Others' category for something special!
```

```
↑ Breads ↑
```

```
BR898 - Banana Bread  
Price: RM 4.50  
Best Before: 17-08-2024  
Allergen: R  
👉 Rich in vitamin A and perfect for healthy treat
```

```
CH123 - Cheezy Bun  
Price: RM 5.00  
Best Before: 27-09-2024  
Allergen: Cheese, Milk  
👉 Extra gooey cheese for cheese aficionado
```

```
GR013 - Garlic Bun  
Price: RM 5.60  
Best Before: 06-11-2024  
Allergen: Garlic  
👉 A soft, fluffy bun infused with rich garlic butter for a savory, aromatic bite.
```

```
.....
```

```
↑ Pastries ↑
```

```
PA932 - Apple Pie  
Price: RM 5.00  
Best Before: 04-09-2024
```

```
PA325 - Bello Minion Biscuit  
Price: RM 3.00  
Best Before: 09-04-2024
```

Figure 4.5.2.2 Sample output of product menu

Products are organized by category, such as Breads, Pastries, Cakes and so on. Additionally, each product is displayed in pairs for easy comparison and readability, with essential details like name, price, expiry date, allergen and brief description. This output style makes the product menu look organized and easy to browse.

4.5.3 Discount Management

```
152 def manage_discounts():
153     while True:
154         print('\n-----')
155         print('\t\t\t', 'DISCOUNT MANAGEMENT')
156         print('-----')
157         print("1. Add Discount")
158         print("2. Modify Discount")
159         print("3. Delete Discount")
160         print("4. View All Discounts")
161         print("5. Exit")
162
163         choice = input("Enter your choice (1-5): ")
164
165         if choice == '1':
166             add_discount()
167         elif choice == '2':
168             modify_discount()
169         elif choice == '3':
170             delete_discount()
171         elif choice == '4':
172             display_discounts()
173         elif choice == '5':
174             confirm_exit = input("Are you sure you want to exit? yes=y / no=n: ").lower()
175             if confirm_exit == 'y':
176                 print("*** Exiting Discount Management. Goodbye! ***")
177                 break
178             else:
179                 print("Returning to the menu.")
180         else:
181             print("|▲ Invalid option. Please try again.|")
```

Figure 4.5.3.1 Function of Discount Management Menu

```
44 def add_discount():
45     # Load existing products
46     products = discount_management()
47
48     product_code = input("Enter product code: ")
49
50     # Check if the product code already exists
51     for details in products.values():
52         if details['product_code'] == product_code:
53             print(f"Product code {product_code} already exists. Please use 'Modify Discount' to update the discount.")
54             return # Exit if product code exists
55
56     product_name = input("Enter product name: ")
57
58     # Validate stock input
59     while True:
60         stock_input = input("Enter product stock: ")
61         if stock_input.isdigit(): # Check if the input is a digit
62             stock = int(stock_input)
63             break # Break the loop if valid input
64         else:
65             print("Please enter a valid positive integer for stock.")
66
67     # Validate price input
68     while True:
69         price_input = input("Enter product price: ")
70         try:
71             price = float(price_input)
72             if price >= 0: # Ensure price is non-negative
73                 break # Break the loop if valid input
```

Figure 4.5.3.2 Function of Add Discount

```

103     def delete_discount():
104         product_code = input("Enter product code: ")
105         found = False
106
107         for details in discounts.values():
108             if details['product_code'] == product_code:
109                 found = True
110                 if details['Discount'] != "0%":
111                     details['Discount'] = "0%" # Set the discount to 0%
112                     save_discounts(discounts)
113                     print(f"Product {product_code}'s discount has been deleted and set to 0%.")
114                 else:
115                     print(f"⚠️The discount for product {product_code} is already 0%, no changes made.|")
116                 break
117
118         if not found:
119             print(f"⚠️No discount found for product {product_code}.|")

```

Figure 4.5.3.3 Function of Delete Discount

```

123     def modify_discount():
124         product_code = input("Enter product code: ")
125         found = False
126
127         for details in discounts.values():
128             if details['product_code'] == product_code:
129                 found = True
130                 new_discount = validate_discount_input() # Use the validated input
131                 details['Discount'] = f"{new_discount}%"
132                 # Update the discount field
133                 save_discounts(discounts) # Save the updated discounts to the file
134                 print(f"Discount for product {product_code} updated to {new_discount}%.")
135                 break
136
137         if not found:
138             print(f"⚠️No discount found for product {product_code}. Use 'Add Discount' to create a new discount.|")

```

Figure 4.5.3.4 Function of Modify Discount

```

140     def display_discounts():
141         print('\n-- Current Discounts and Product Details -- ')
142         print('-' * 70)
143         for product_number, product_info in discounts.items():
144             product_name = product_info.get('Product Name', '').ljust(20)
145             product_code_display = product_info.get('product_code', '').ljust(15)
146             stock = str(product_info.get('Stock', 0)).ljust(10)
147             price = product_info.get('Price', '').ljust(15)
148             discount = product_info.get('Discount', '0%').ljust(15)
149             print(f'{product_name}{product_code_display}{stock}{price}{discount}')
150
151         print('-' * 70)

```

Figure 4.5.3.5 Function of Display Discount

The discount management function aims to enable cashiers to manage product discounts in an organized way. First, the ‘discount_management()’ function will loads existing product discount information from a file. The ‘save_discounts()’ function ensures updated discounts for the products can be saved back. Also, the ‘validate_discount_input()’ function checks whether the discount input is between 0 and 100. Furthermore, the ‘add_discount()’ function enable cashier to add discounts for newly launched products by entering product details such as product code, product name, product stock, product price and product discount while ‘modify_discount()’ function allows the cashier to change discounts for existing items. The ‘display_discounts()’ function will presents a clear overview of current discounts for clearer understanding. Lastly, the ‘manage_discounts()’ function provides the cashier with options to add, modify, delete or view current discounts based on their selection.

Sample output:

```
-----  
DISCOUNT MANAGEMENT  
-----  
1. Add Discount  
2. Modify Discount  
3. Delete Discount  
4. View All Discounts  
5. Exit  
Enter your choice (1-5): 1  
Enter product code: PA932  
Product code PA932 already exists. Please use 'Modify Discount' to update the discount.
```

Figure 4.5.3.6 Sample output of Add Discounts for an existing product

```
-----  
DISCOUNT MANAGEMENT  
-----  
1. Add Discount  
2. Modify Discount  
3. Delete Discount  
4. View All Discounts  
5. Exit  
Enter your choice (1-5): 1  
Enter product code: AB123  
Enter product name: pizza  
Enter product stock: 20  
Enter product price: 15  
Enter discount percentage (0-100): 10  
New product pizza added with a discount of 10.0 %.
```

Figure 4.5.3.7 Sample output of Add Discounts for a new product

```
-----  
DISCOUNT MANAGEMENT  
-----  
1. Add Discount  
2. Modify Discount  
3. Delete Discount  
4. View All Discounts  
5. Exit  
Enter your choice (1-5): 2  
Enter product code: CH123  
Enter discount percentage (0-100): 10  
Discount for product CH123 updated to 10.0%.
```

Figure 4.5.3.8 Sample output of Modify Discounts

```
-----  
DISCOUNT MANAGEMENT  
-----  
1. Add Discount  
2. Modify Discount  
3. Delete Discount  
4. View All Discounts  
5. Exit  
Enter your choice (1-5): 3  
Enter product code: CH123  
Product CH123's discount has been deleted and set to 0%.
```

Figure 4.5.3.9 Sample output of Delete Discounts

```

-----  

DISCOUNT MANAGEMENT  

-----  

1. Add Discount  

2. Modify Discount  

3. Delete Discount  

4. View All Discounts  

5. Exit  

Enter your choice (1-5): 4  

-- Current Discounts and Product Details --  

-----  

tiramisu      CK858      1      RM 8.50      0.0%  

cheezy bun    CH123      2      RM 5.00      10.0%  

banana bread  BR890      1      RM 4.50      0.0%  

red velvet    CK857      1      RM 10.50     0.0%  

apple pie     PA932      1      RM 5.00      0.0%  

gingerbread   BS678      2      RM 2.50      10.0%
-----
```

Figure 4.5.3.10 Sample output of View All Discounts

```

-----  

DISCOUNT MANAGEMENT  

-----  

1. Add Discount  

2. Modify Discount  

3. Delete Discount  

4. View All Discounts  

5. Exit  

Enter your choice (1-5): 5  

Are you sure you want to exit? yes=y / no=n: Y  

*** Exiting Discount Management. Goodbye! ***
-----
```

Figure 4.5.3.11 Sample output of Exit Discount Management

In the cashier discount management system, the cashier can select option 1 to add a discount for a new product by entering product details as shown in figure 4.5.3.7. Based on figure 4.5.3.6, they cannot add discounts for products that already exist as they need to select option 2 to modify existing product discounts. Additionally, the cashier can delete a discount by selecting option 3 and entering the product code as shown in figure 4.5.3.9. Based on the figure 4.5.3.10, cashier is allowed to view all discounts at once by selecting option 4. Finally, the cashier can exit the discount management system by selecting option 5 as shown in figure 4.5.3.11.

4.5.4 Transaction Completion

```
155 def manual_generate_receipt():
156
157     while True:
158         customer_info = load_data_from_customer_order_list()
159
160         print('\n❗ The receipt will be generated automatically after the customers complete their purchase.')
161         print('❗ The receipt can also be manually generated if the customers request it for their previous order.')
162         manual_generate = input('\nDo you want to generate the receipt manually? (y=yes, n=no)\n>> ')
163
164         if manual_generate not in ['y', 'n']:
165             print('\n+-----+')
166             print('⚠ Invalid input. Please enter again. !')
167             print('+-----+')
168
169         elif manual_generate == 'y':
170             while True:
171                 index = 1
172                 print('\n', '-' * 47)
173                 centered(word='ORDER LIST', width=47)
174                 print('-' * 47)
175                 for cart_id, value in customer_info.items():
176                     print(f'{index}. {cart_id} - {value["order_id"]}')
177                     index += 1
178
179                 order_receipt = input('\nPlease enter the cart ID: ')
180                 if order_receipt not in customer_info.keys():
181                     print('\n+-----+')
182                     print('⚠ Cart ID not found. Please enter again. !')
183                     print('+-----+')
184                     continue
185                 else:
186                     receipt(customer_info)
```

Figure 4.5.4.1 Function to generate receipt manually

```
49 def receipt(customer):
50     customer_info = load_data_from_customer_order_list() # store the data that retrieved from file into customer_info
51     print(' ')
52     # header of the receipt
53     header = ['MORNING GLORY BAKERY',
54               'Reg: 219875000123 (86851-Z)',
55               '51, Lorong Maplewood, Taman Springfield, 47180, Puchong, Selangor.',
56               'Tel: 04-5678951',
57               'Email: morningglorybakery@gmail.com',
58               'Business hours: 9.00AM - 6.00PM']
59
60     print('')
61     print('-' * terminal_width)
62     for i in header:
63         centered(i, terminal_width) # call the function to print the header in the middle
64
65     print('\n' + '=' * terminal_width) # print a separate line
66     current_time = datetime.now().strftime("%I:%M:%S %p") # convert the time into string for readability and store it into "current_time"
67     current_date = datetime.now().strftime("%d-%m-%Y") # convert the date into string for self-formatting and store it into "current_date"
68     print(current_time + current_date.rjust(terminal_width-len(current_date)-1)) # specify the format of printing date and time
69     print(' ') # blank a line
70     receipt_no = random.randint(a=1000001, b=10000000) # determine the invoice number in random in a range from 1000001 to 10000000
71     print(f'{"Receipt no":<11}: MGB-{str(receipt_no)}')
72
73     for cart_id, customer_details in customer_info.items():
74         if customer in cart_id:
75             username = customer_details['username']
76             order_id = customer_details['order_id']
```

Figure 4.5.4.2 Function to generate receipt for customers

In figure 4.5.4.1, cashier can generate the receipt manually if customers request it for their previous orders. The system will ask the cashier whether she wants to generate the receipt manually or not. If the cashier insists to do so, she will need to enter the cart id from the given

list to proceed to the function in figure 4.5.4.2. The function in figure 4.5.4.2 will generate a well-designed receipt for customers. It has a parameter, which is customer, to allow it can generate a receipt for customers based on their cart id. Firstly, it will read the data from customer order list file. A header of the receipt is displayed followed by the current time and current date that the receipt is generated. The time is in the format of ‘hours: minutes: seconds AM/PM’, while the date is in the format of ‘day-month-year’. ‘rjust()’ function is used for the design purpose, positioning the date at the rightmost in the defined terminal width, 100 characters.

The receipt number is generated using random.randint() function to generate a random integer number for every receipt. It will also retrieve the username, order id, and cart id of customers to display in the receipt as a proof. It collects the quantity and items’ price to calculate the subtotal to be paid by customers. If the product bought by the customer have discounted price, the system will retrieve the discounted price from the file, else the price remains as usual. After summing up all the total, the total must be multiplied with 6% as there is a service tax of 6% for every purchase. The receipt will display the points earned by the customers for the purchase in the same line with the total. An appreciation and warning (Goods sold are not refundable and returnable) are printed at the bottom of the receipt. Lastly, a record of each receipt generated is saved to file for future reference.

Sample output:

```
💡 The receipt will be generated automatically after the customers complete their purchase.  
💡 The receipt can also be manually generated if the customers request it for their previous order.  
  
Do you want to generate the receipt manually? (y=yes, n=no)  
>>> y  
  
-----  
          ORDER LIST  
-----  
1. 5580946551 - ORD001  
2. 5684950188 - ORD002  
3. 1544631417 - ORD003  
4. 4418942694 - ORD004  
5. 7841554976 - ORD005  
6. 5548156914 - ORD006  
7. 9875428598 - ORD007  
8. 4567891234 - ORD008  
9. 6543219876 - ORD009  
10. 3216549870 - ORD010  
11. 1597534862 - ORD011  
12. 7896541230 - ORD012  
13. 2345678901 - ORD013  
14. 8765432109 - ORD014
```

Figure 4.5.4.3 Sample output of the cashier decides whether to generate receipt manually or not

```

-----  

MORNING GLORY BAKERY  

Reg: 219875000123 (80851-Z)  

51, Lorong Maplewood, Taman Springfield, 47180, Puchong, Selangor.  

Tel: 04-5678951  

Email: morningglorybakery@gmail.com  

Business hours: 9.00AM - 6.00PM  

=====  

03:01:28 PM           26-10-2024  

Receipt no : MGB-5481360  

Username   : chew  

Order ID   : ORD044  

Cart ID    : 4992374927  

-----  

Item          Qty       Price(RM)      Amount(RM)  

-----  

Cheezy Bun     2         5.00          10.00  

Gingerbread    2         2.50          5.00  

-----  

Subtotal:      15.00  

Discounted price: 13.50  

Service tax @ 6%: 0.81  

Points earned: 135          TOTAL:        14.31  

-----  

Goods sold are not returnable and refundable !  

***** Thank You Please Come Again *****  

-----
```

Figure 4.5.4.4 Sample output of a generated receipt

Figure 4.5.4.3 shows the cashier prefer to generate the receipt manually. A list of cart id and order id is provided, the cashier will enter the cart id to generate a receipt. Figure 4.5.4.4 demonstrates a well-designed receipt that is generated for customers after they paid for their purchase including all the details.

4.5.5 Reporting

```
91 # define the function to generate sales performance report and product popularity report
92 # usages ± hxlum
93 def generate_sales_report():
94     print('')
95     printed_centered('SALES PERFORMANCE REPORT') # display type of sales report user can generate
96     print('1. Yearly report')
97     print('2. Monthly report')
98     print('3. Product popularity')
99     print('4. Back to previous page')
100    while True:
101        # collect the choice of user and execute corresponding functions
102        report_service = input('\nPlease input the index number of the service you choose: ')
103
104        if validation_empty_entries(report_service):
105            if report_service == '1':
106                yearly_sales_performance()
107                break
108            elif report_service == '2':
109                monthly_sales_performance()
110                break
111            elif report_service == '3':
112                product_popularity()
113                break
114            elif report_service == '4':
115                return False
116            else:
117                print('\n+-----+')
118                print('|\u25bc Please enter a valid index number. |')
119                print('+-----+')
```

Figure 4.5.5.1 Function of sales performance report options

The function displays a menu option for generating different types of sales reports. It begins by printing the title ‘SALES PERFORMANCE REPORT’ at centre, and a list of report options for user to select. A while loop is used to ensure only valid input is processed. If user entered ‘1’, yearly_sales_performance() function is called, if user entered ‘2’, monthly_sales_performance() function is called, if user entered ‘3’, product_popularity() function is called, and if user entered ‘4’, they will exit the function by returning False, and return to baker privilege page. If the input is not digit and falls under valid range, user is prompted to enter again. The function validation_empty_entries() is also used to detect whether the input have value. The loop continues until a valid choice is made.

Sample Output:

```
-----  
SALES PERFORMANCE REPORT  
-----  
1. Yearly report  
2. Monthly report  
3. Product popularity  
4. Back to previous page  
  
Please input the index number of the service you choose: d  
  
+-----+  
| ▲ Please enter a valid index number. |  
+-----+  
  
Please input the index number of the service you choose: #  
  
+-----+  
| ▲ Please enter a valid index number. |  
+-----+  
  
Please input the index number of the service you choose: 5  
  
+-----+  
| ▲ Please enter a valid index number. |  
+-----+  
  
Please input the index number of the service you choose: |
```

Figure 4.5.5.2 Sample output of sales report menu

Based on the output, a clear title is printed at the beginning following with a list of sales performance options. If user enter a valid input, corresponding function will be called. If user enter invalid input, they will be prompted again until valid input is detected.

(a) Identify allowable years and months

```
121 # define the function to find out the available years for users to choose based on previous customer payment history  
122 # usages : hxlum  
123 def allowable_year():  
124     allowable_years = set()  
125     for key, items in cashier_transactionKeeping.items():  
126         order_date = datetime.strptime(items['order_date'], '%d-%m-%Y')  
127         allowable_years.add(order_date.year)  
128  
129     allowable_years_unpack = ', '.join(str(year) for year in list(allowable_years))  
130  
131     return allowable_years_unpack  
132  
133 # define the function to find out the available months for users to choose based on previous customer payment history  
134 # usages : hxlum  
135 def allowable_month(report_year):  
136     allowable_months = set()  
137     for key, items in cashier_transactionKeeping.items():  
138         order_date = datetime.strptime(items['order_date'], '%d-%m-%Y')  
139         if order_date.year == int(report_year): # identify available months based on particular year selected by user  
140             allowable_months.add(order_date.month)  
141  
142     allowable_month_unpack = ', '.join(str(month) for month in list(allowable_months))  
143  
144     return allowable_month_unpack
```

Figure 4.5.5.3 Functions that identify allowable years and months

The first function (allowable_year()) is used to identify the allowable years available for sales report generation based on transaction data stored in cashier transaction keeping file. This function retrieves all unique years from past transactions (order date) using sets. It loops through each transaction data, converting the order_date string into a date object, and extract the year component.

The allowable_month(report_year) function works similarly to allowable_year() function, but it have a parameter, report_year. This function will also loop through each transaction data in the file, converting the order_date string into a date object, but only extracting the exist month when the year matched the selected year (report_year). If the year matched, it extracts the month and adds it into the set.

Sample output:

```
Allowable year: 2024, 2023  
Please enter the year you want to generate report: 2024  
  
Allowable month: 1, 2, 5, 6, 7, 8, 9, 10  
Please enter the month you want to generate report: 9
```

Figure 4.5.5.4 Sample output that display allowable years and months

Based on the output, users can select valid years and months for following process, avoiding errors and inaccurate data being calculated.

(b) Yearly sales performance report

```
146 # define the function to generate yearly sales report
147 #usages ±hxium+1
148 def yearly_sales_performance():
149     while True:
150         while True:
151             # generate a 4 digit unique report number for each report
152             report_num = random.randint(a:1000, b:9999)
153             if report_num not in sales_report_file.keys(): # break when the generated report number is not duplicate
154                 break
155
156             allowable_years = allowable_year() # determine available years
157
158             print(f'\nAllowable year: {allowable_years}') # print available years
159
160             report_year = input('Please enter the year you want to generate report: ') # collect user selected year
161             if validation_empty_entries(report_year):
162                 if report_year in allowable_years: # check if selected year is in the list of valid years
163
164                     # initialize counter for sales and orders
165                     total_sales = 0
166                     total_orders = 0
167                     previous_total_sales = 0
168                     previous_total_orders = 0
169                     previous_year = int(report_year) - 1 # calculate the previous year for comparison
170
171                     # loop through customer payment history
172                     for key, item in cashier_transactionKeeping.items():
173                         order_date = datetime.strptime(item['order_date'], __format: '%d-%m-%Y')
174                         order_year = order_date.year
175                         if order_year == int(report_year): # if met the same year as the selected year
176                             total_sales += float(item['total_spend(RM)']) # add the total spend
```

Figure 4.5.5.5 Function that generate yearly sales report

The function begins by generating a unique 4 digits report number, and ensure no duplicate founded in cashier sales report file before proceeding. It then call allowable_year() function to display allowable years and let user to select. If the entered year is valid, the function initialized counters, total_sales, total_orders, previous_total_sales and previous_total_orders to 0, for performance comparison between selected year and prior year.

As the function iterate through transaction data, it will check the order year against the selected and previous years. If the order year matches one of the selected years or previous year, it will add the total spend of customer to total_sales (or previous_total_sales) and increment total_orders (or previous_total_orders) by 1. After finish looping, it will calculate the sales growth ratio and percentage difference between both years, if there are sales records in both

years. If no, it will display ‘No previous sales’ message. It then prints the summary including total sales, total orders, sales growth ratio and actual percentage growth.

Last, it will prompt user whether to save the report. If user choose to save, it will first check the cashier sales report file to ensure there are no duplicate same type and same year report. If no duplicate detected, it would save data such as report type, selected years, total sales and total orders to cashier sales report file. The function will then ask if the user would like to generate another report. If yes, it recursively calls itself, if no, user will return to the sales report menu.

Sample Output:

```
2. Monthly report
3. Product popularity
4. Back to previous page

Please input the index number of the service you choose: 1

Allowable year: 2024, 2023
Please enter the year you want to generate report: 2024

-----
          2024'S YEARLY PERFORMANCE SUMMARY
-----

Total Sales : 2789.68
Total Orders : 51

Sales Growth(%) for 2024 Compared to 2023 : 1766.29%
Actual Percentage Growth                 : 1666.29%

Save this report to system? (y=yes, n=no): y

+-----+
|⚠ The report cannot be saved because the same report has been saved previously.|+
+-----+

Continue generating yearly sales report? (y=yes, n=no)
>>> n

Exit the yearly performance report page.....
```

Figure 4.5.5.6 Sample output for yearly sales report (I)

```

Allowable year: 2024, 2023
Please enter the year you want to generate report: 2023

-----
2023'S YEARLY PERFORMANCE SUMMARY
-----

Total Sales : 157.94
Total Orders : 11

Sales Growth(%) for 2023 Compared to 2022 : No previous sales.
Actual Percentage Growth : No previous sales.

Save this report to system? (y=yes, n=no):

```

Figure 4.5.5.7 Sample output for yearly sales report (2)

Figure 4.5.5.6 shows that the function will accurately calculate the total sales and total orders based on user selected years. Besides, it will also calculate the total sales of previous year for performance comparison. Figure 4.5.5.7 shows that when there are no sales data in previous year, it will display the message correctly.

(c) Monthly sales performance report

```

264     # define function to generate monthly sales report
265     2 usages + hxlum+1
266     def monthly_sales_performance():
267         while True:
268             while True:
269                 # generate a 4 digit unique report number for each report
270                 report_num = random.randint( a: 1000, b: 9999)
271                 if report_num not in sales_report_file.keys(): # break when the generated report number is not duplicate
272                     break
273
274             allowable_years = allowable_year() # determine available years
275
276             print(f'\nAllowable year: {allowable_years}')
277
278             # collect user selected year
279             report_year = input('Please enter the year you want to generate report: ')
280             if validation_empty_entries(report_year):
281                 if report_year in allowable_years: # check if selected year is in the list of valid years
282                     break
283                 else:
284                     print('\n+-----+')
285                     print('!▲ Please enter a valid year based on given year. |')
286                     print('+-----+')
287
288             while True:
289                 allowable_months = allowable_month(report_year) # determine available months
290
291                 print(f'\nAllowable month: {allowable_months}')
292
293                 # collect user selected month
294                 report_month = input('Please enter the month you want to generate report: ')
295                 if validation_empty_entries(report_month):

```

Figure 4.5.5.8 Function to generate monthly sales performance report

This function works similarly to the yearly_sales_performance() function. The function begins by generating a unique 4 digits report number, and ensure no duplicate founded in cashier sales report file before proceeding. It then call allowable_year() function to display allowable years and let user to select. If the entered year is valid, it will call allowable_month() function to let user select their desired month. If the month is valid, the function initialized counters, total_sales, total_orders, previous_total_sales and previous_total_orders to 0, for performance comparison between selected month and prior month.

As the function iterate through transaction data, it will check the order year against the selected and previous years, and then check the order month against the selected and previous month. If the order year and month matches one of the selected years and month or previous year and month, it will add the total spend of customer to total_sales (or previous_total_sales) and increment total_orders (or previous_total_orders) by 1. After finish looping, it will calculate the sales growth ratio and percentage difference between both months, if there are sales records in both months. If no, it will display ‘No previous sales’ message. It then prints the summary including total sales, total orders, sales growth ratio and actual percentage growth.

Last, it will prompt user whether to save the report. If user choose to save, it will first check the cashier sales report file to ensure there are no duplicate same type and same year and month report. If no duplicate detected, it would save data such as report type, selected years, selected month, total sales and total orders to cashier sales report file. The function will then ask if the user would like to generate another report. If yes, it recursively calls itself, if no, user will return to the sales report menu.

Sample Output:

```
Please input the index number of the service you choose: 2

Allowable year: 2024, 2023
Please enter the year you want to generate report: 2024

Allowable month: 1, 2, 5, 6, 7, 8, 9, 10
Please enter the month you want to generate report: 9

-----
9/2024 SALES PERFORMANCE SUMMARY
-----
Total Sales      : 71.55
Total Orders     : 2

Sales Growth for 9/2024 Compared to 8/2024 : 27.78%
Actual Percentage Growth                   : -72.22%

Save this report to system? (y=yes, n=no): y
Successfully saved!

Continue generating monthly sales report? (y=yes, n=no)
>>> y

Allowable year: 2024, 2023
Please enter the year you want to generate report: |
```

Figure 4.5.5.9 Sample Output of monthly sales performance report

```
14    },
15    "5278": {
16        "report_type": "monthly sales report",
17        "selected_year": "2024",
18        "selected_month": "9",
19        "total_sales": 71.55,
20        "total_order": 2
21    }
22 }
```

Figure 4.5.5.10 Sample Output of monthly report saved to text file

The output shows that the function will accurately calculate the total sales and total orders based on user selected years and month. Besides, it will also calculate the total sales of previous year and month for performance comparison. It will also save the report to cashier sales report file if no duplicate report is found.

(d) Product Popularity Report (customer rating and review)

```
404     # define the function to calculate the product popularity based on users option
405     3 usages  ▾ hxlum
406
407     def product_popularity_customer_review(report_year, report_month, best_seller, least_seller, max_quantity, min_quantity,
408                                                 all):
409
410         # initialize variables to store ratings and reviews for best and least seller product
411         best_rating = 0
412         best_count = 0
413         best_review = {}
414         least_rating = 0
415         least_count = 0
416         least_review = {}
417
418         # loop through customer reviews
419         for key, item in customer_review.items():
420             # extract the year and month of each review
421             order_date = datetime.strptime(item['order_date'], '%d-%m-%Y')
422             order_year = order_date.year
423             order_month = order_date.month
424
425             # if user choose to generate overall product popularity report
426             if all:
427                 # calculate the sum of rating and add the corresponding reviews to best_review dictionary
428                 if best_seller in item['product_name']:
429                     best_rating += item['rating']
430                     best_count += 1
431                     best_review[item['username']] = item['review']
432
433                 # calculate the sum of rating and add the corresponding reviews to least_review dictionary
434                 if least_seller in item['product_name']:
435                     least_rating += item['rating']
436                     least_count += 1
437                     least_review[item['username']] = item['review']
```

Figure 4.5.5.11 Function that retrieve customer rating and review for selected year or month

The product popularity report is divided into two parts. The first part is to analyze the product with best sales and least sales, while the second part is to calculate the average ratings and extract customer reviews for both products. This function analyses customer review and ratings over a chosen time period, either overall, annually or monthly.

First, the function initializes variables to store rating, review counts and actual review for both best-selling and least-selling products. It then loops through all customer review in customer review file, parsing each review's date to extract the year and month. The required input of the function depends on the type of report user selected. If user decided to generate yearly product popularity report, the input of selected year, name of best seller and least seller is required to detect the accurate customer ratings and review. If user decided to generate monthly product popularity report, the input of selected year, selected month and name of best seller and least seller is required. For an overall report (all parameter equal to True), it sums ratings and gather reviews for the best and least-selling products across all dates.

If the best-selling and least-selling products are the same (usually happens when there is only one product across specific time period), the function duplicates the best-seller data for both

categories. After filtering, if there are rating or review for either product, it calculates the average ratings and selects a random review with the customer username. If there is not any rating for the product, it returns the message ‘no rating detected’ and ‘no review detected’. Last, the function returns the average rating, a random review and the corresponding customer username (if found) of both products.

Sample output:

```
 ❶ Top-selling product      : Cheezy Bun
 ❷ Total quantity sold     : 206
 ❸ Average customer rating : 4.0
 ❹ Customer review spotlight : "A soft, fluffy bun filled with a rich, melted cheese, nice!" ----- nct

 ❺ Least-selling product    : Red Velvet
 ❻ Total quantity sold      : 5
 ❼ Average customer rating   : 4.0
 ❽ Customer review spotlight : "good" ----- Mark0825
```

Figure 4.5.5.12 Sample output of average rating and random customer review

Based on the output, user can gain insight of customer feedback trends over time, and help tracking customer preferences in a specific time period.

(e) Product Popularity Report (best-seller and least-seller)

```
492 # define the function to calculate and display product popularity based on user option
493 usage = nxlum
494 def product_popularity():
495     while True:
496         while True:
497             # display option for report type
498             print('')
499             printed_centered('PRODUCT POPULARITY')
500             print('1. Yearly report')
501             print('2. Monthly report')
502             print('3. Overall product popularity summary')
503             print('4. Back to previous page')
504             report_service = input('\nPlease input the index number of the service you choose: ')
505             if validation_empty_entries(report_service):
506
507                 # if user choose to generate yearly report
508                 if report_service == '1':
509                     allowable_years = allowable_year()
510
511                     print(f'\nAllowable year: {allowable_years}')
512
513                     report_year = input('Please enter the year you want to generate report: ')
514                     if validation_empty_entries(report_year):
515                         if report_year in allowable_years: # ensure the input is within available years
516                             product_ordered = {}
517                             # calculate quantity sold for each product in the selected year
518                             for key, item in cashier_transactionKeeping.items():
519                                 order_date = datetime.strptime(item['order_date'], __format: '%d-%m-%Y')
520                                 order_year = order_date.year
521                                 if order_year == int(report_year):
522                                     for product in item['items']:
```

Figure 4.5.5.13 Function that calculate the best-selling and least-selling products

This function is the first part of product popularity report, which carry out analysis about the best-selling and least-selling products over a specific period. First, the function displays a title ‘PPRODUCT POPULARITY’ in the centre, with a list of options: yearly, monthly or overall product popularity reports. When the user selects a report type, their input is validated to ensure entries are not empty and fall within valid ranges.

If user choose ‘1’, the function will call allowable_year() function to loop through transaction history to find out allowable year. The user then enters a specific year, which the function will check against allowable years. Next, it will sum up the total quantity sold per product to identify the best and least-selling products. Next, product_popularity_customer_review() function is called to fetch the average rating and a random review of both products. Last, a summary of the top and least-sold product id printed, including product name, total quantity sold, average ratings and spotlight review if available.

If user choose ‘2’, the function will call allowable _month() function to loop through transaction history to find out allowable year and month. The process is similar as generating yearly product popularity report, is just user need to enter a year and month, and the best and least-selling product will be detected and calculated based on that year and month. Same as customer rating and reviews, the function will loop through customer review data and focus on reviews with order date same as the selected year and month. Last, the function identifies the best and least selling products, retrieves their average ratings and reviews, and print a summary of monthly product popularity report.

If user choose ‘3’, the function will calculate quantity sold of all products across all transaction records without limiting the year and month, identify the top and least-selling products, fetched their average rating and reviews, and print the overall summary. If user choose ‘4’, they will exit the loop and return to baker privilege page. User are prompted to enter again if invalid input is detected.

Sample Output:

```
-----  
          PRODUCT POPULARITY  
-----  
1. Yearly report  
2. Monthly report  
3. Overall product popularity summary  
4. Back to previous page  
  
Please input the index number of the service you choose: 2  
  
Allowable year: 2024, 2023  
Please enter the year you want to generate report: 2024  
  
Allowable month: 1, 2, 5, 6, 7, 8, 9, 10  
Please enter the month you want to generate report: 10
```

Figure 4.5.5.14 Sample output of prompting user to select report type

```
-----  
10/2024 PRODUCT POPULARITY SUMMARY  
-----  
  
❶ Top-selling product      : Tiramisu  
❷ Total quantity sold     : 168  
❸ Average customer rating  : No rating detected.  
❹ Customer review spotlight : No review detected.  
  
❺ Least-selling product    : Red Velvet  
❻ Total quantity sold      : 5  
❼ Average customer rating   : 5.0  
❽ Customer review spotlight : "good" ----- Mark0825
```

Figure 4.5.5.15 Sample output of corresponding product popularity report

The output displays a summary of product popularity based on user-selected criteria (yearly, monthly, or overall). Each report will display the best and least-selling products name, total quantity sold, average rating and review spotlight if available.

4.6 Baker(s)

4.6.1 Sign Up/ Log In

```
44 # baker account login function
45 #usage : hxlm+1
46 def baker_accounts():
47
48     print('')
49     printed_centered('BAKER')
50     baker_username = input('Username: ') # ask for username
51     if baker_username not in baker_info: # continue looping if username not match with the name
52         print('\n+-----+')
53         print('!▲ You are not a baker, cannot access to baker privilege. !')
54         print('+-----+\n')
55         print('Exiting to Main Page.....')
56         return False
57
58     else:
59         baker_password = input('Password: ') # ask for baker's password
60         while len(baker_password) < 8 or len(baker_password) > 12: # repeating the prompt to input when password length
61             print('\n+-----+')
62             print('!▲ Invalid password length. Please make sure it is between 8 to 12 digits! !')
63             print('+-----+\n')
64             baker_password = input('Password: ')
65
66         while baker_password != baker_info[baker_username]['baker_password']: # continue to validate the password of ba
67             print('\n+-----+')
68             print('!▲ Incorrect password. Please enter again. !')
69             print('+-----+\n')
70             baker_password = input('Password: ')
71
72         print('\nLogin successfully!') # login successfully if the password meet the 2 requirements above
73         print(f"Welcome, baker {baker_username}!")
74
75     while True:
```

Figure 4.6.1.1 Function of baker login or sign in

The function begins by prompting user to enter baker username. Next, it will validate user input by matching it with recorded baker username in baker file. If the function unable to match the user input with baker username stored in baker file, it will exit the loop and return to the main page. If matches, it will then prompt user to enter password and match the input again with the password of user selected baker stored in baker text file. If the password matches, it will display the baker privilege menu with options for managing recipes, checking inventory, keeping product records, manging equipment, and return to main page. Corresponding function is called based on user input. If user enter an invalid input, it will display error message and prompt user again.

Sample Output:

```
-----  
BAKER  
-----  
Username: lum  
+-----+  
|⚠ You are not a baker, cannot access to baker privilege. |  
+-----+  
  
Exiting to Main Page.....  
  
Role Options:  
1. Manager 🏠👤  
2. Baker 🍞👤  
3. Cashier 💼👤  
4. Customer 🧑🏼👤  
5. Exit the program 🔁➡  
  
Who are you? (1, 2, 3, 4):  
>>> |
```

Figure 4.6.1.2 Sample output of invalid baker username

```
-----  
BAKER  
-----  
Username: christine  
Password: 23456789  
  
Login successfully!  
Welcome, baker christine!  
  
-----  
BAKER PRIVILEGE  
-----  
1. Recipe management  
2. Inventory check  
3. Product record-keeping  
4. Equipment management  
5. Back to Main page  
  
Select a choice (1, 2, 3, 4, 5):  
>>>
```

Figure 4.6.1.3 Sample output of valid baker username and password

Figure 4.6.1.2 shows the scenario when the function cannot match username input by user with any recorded baker data. Figure 4.6.1.3 shows the scenario when user enter a valid baker username and password, where the function will display the baker privilege menu and call the function based on user input options.

4.6.2 Equipment Management

```
130     # define function that let user to select equipment management option
131     3 usages  ▲ hxlum +1
132     def equipment_management():
133         # display the equipment management option
134         print('')
135         printed_centered('EQUIPMENT MANAGEMENT MENU')
136         print('1. Report Malfunction')
137         print('2. Report Maintenance Needs')
138         print('3. Back to Homepage\n')
139
140         # loop until a valid choice is selected or user choose to exit
141         while True:
142             equipment_management_type = input('Please choose a service:\n'
143                                              '    >>> ').strip()
144             if validation_empty_entries(equipment_management_type):
145                 if equipment_management_type == '1':
146                     equipment_malfunction()
147                 elif equipment_management_type == '2':
148                     equipment_maintenance()
149                 elif equipment_management_type == '3':
150                     print('Exiting to baker privilege...')
151                     break
152                 else:
153                     print('\n+-----+')
154                     print('|▲ Please enter a valid service type. |')
155                     print('+-----+\n')
```

Figure 4.6.2.1 Function of Equipment Management Menu

The function above displays a menu for users to manage any issues related to bakery equipment. First, it will display a header, “EQUIPMENT MANAGEMENT MENU”, and provide three options for users to choose, including report malfunctions, report maintenance needs and return to homepage. Based on the user’s choice, the corresponding function is called. If user chooses ‘1’, equipment_malfunction() is called to handle malfunction reports, if user chooses ‘2’, equipment_maintenance() to handle maintenance report and if they chooses ‘3’, they exit to the baker privilege page. A while loop is used to continue prompting the user until a valid input is provided. For example, the validation_empty_entries() function ensures the input have a value. If the input is empty, an error message is printed, and user is prompt to enter again.

Sample output:

```
-----  
EQUIPMENT MANAGEMENT MENU  
-----  
1. Report Malfunction  
2. Report Maintenance Needs  
3. Back to Homepage  
  
Please choose a service:  
>>> d  
  
+-----+  
| ▲ Please enter a valid service type. |  
+-----+  
  
Please choose a service:  
>>> #  
  
+-----+  
| ▲ Please enter a valid service type. |  
+-----+  
  
Please choose a service:  
>>>
```

Figure 4.6.2.2 Sample output of Equipment Management Menu

Figure 4.6.2.2 shows that user is able to make clear and informed decisions by viewing the instructions, and the system also ensure that the input is valid before heading to next step. This validation helps prevent errors and provides a smooth and guided experiences.

```

157 # define function that let user to selected which equipment to report
158 usages += hulum+1
159 def equipment_lists():
160     print('')
161     printed_centered('EQUIPMENT LIST')
162     index = 1 # initialize index for each equipment
163     equipment_index_mapping = {} # create a dictionary to map equipment index with its category and details
164
165     # loop through each equipment category and display equipment names with corresponding index
166     for category, items in equipment_category_groups.items():
167         print(f' {category} ')
168         for equipment in items:
169             print(f'{index}. {equipment["equipment_name"].title()}')
170             equipment_index_mapping[index] = (category, equipment) # map index to category and equipment details
171             index += 1
172     print('')
173
174     while True:
175         # get user selected equipment
176         selected_index = input('Please select the equipment you want to report. (enter the number of equipment)\n'
177                               '">>>> ').strip()
178         if validation_empty_entries(selected_index):
179             try:
180                 selected_index = int(selected_index) # if input is a digit and fall within valid range
181                 if 1 <= selected_index <= len(equipment_index_mapping):
182                     break # exit loop when valid selection is made
183                 else:
184                     # if input is invalid, prompt user to enter again
185                     print('\n+-----+')
186                     print('!▲ Please enter a valid number based on list given. |')
187                     print('+-----+\n')
188             except ValueError:

```

Figure 4.6.2.3 Function that group and display equipment by category

The function first prints the title “EQUIPMENT LIST”, and then initialize the index counter to 1 to label each equipment. A dictionary named ‘equipment_index_mapping’ is created to match each equipment’s category and details to their corresponding index. The function then loops through equipment_category_groups, which is a dictionary that groups all equipment by their category. The category, equipment name and index are then printed for user to choose.

After displaying the equipment list, the function enters a loop to get the index of user selected equipment. If the input is invalid or empty, an error message is displayed, and user is prompted to enter again. If the input is valid, the function will return a tuple containing the selected index and its mapped category and equipment details.

Sample Output:

```
-----  
EQUIPMENT LIST  
-----  
↑ Ovens and Baking Appliances ↑  
1. Convection Oven  
2. Toaster Oven  
  
↑ Mixing Equipment ↑  
3. Stand Mixer  
  
Please select the equipment you want to report. (enter the number of equipment)  
>>>  
  
! Please enter something...  
  
Please select the equipment you want to report. (enter the number of equipment)  
>>> #  
  
+-----+  
| ▲ Please enter a valid number. (Cannot contain spacing.) |  
+-----+  
  
Please select the equipment you want to report. (enter the number of equipment)  
>>> d  
  
+-----+  
| ▲ Please enter a valid number. (Cannot contain spacing.) |  
+-----+
```

Figure 4.6.2.4 Sample output of equipment list

The output shows that user is able to make clear and informed decisions by viewing the equipment list provided, and the system also ensure that the input is valid before proceeding. This validation helps prevent errors and provides a smooth and guided experiences.

(a) Malfunction report

```
195     # define function that let user to make malfunction equipment report
196     2 usages  ▲ hxlum+1
197     ✓ def equipment_malfunction():
198         malfunction_data = load_data_from_manager_notifications()  # load previous malfunction report from file
199         print('')
200         print('-' * 140)
201         print('\nWelcome to the Equipment Malfunction Report page. Please follow the instructions to report any issues.')
202         selected_equipment, equipment_info = equipment_lists()  # get user selected equipment and its information
203         category, equipment = equipment_info # unpack the tuple to get category and other details
204
205         # display the details of selected equipment
206         max_length = len('Next Scheduled Maintenance')
207
208         print('\nBasic details of selected equipment:\n')
209         print(f'{"Category":<28}: {category.title()}') # category
210         print(f'{"Equipment Name":<28}: {equipment["equipment_name"].title()}') # equipment name
211         index = 1
212         for serial_number in equipment['serial_number']:
213             print(f'Serial Number {str(index).ljust((max_length + 1) - len("Serial Number"))}: {serial_number}')
214             index += 1
215         print(f'{"Manufacturer":<28}: {equipment["manufacturer"].title()}') # manufacturer
216         print(f'{"Model Number":<28}: {equipment["model_number"]}') # model number
217         print(f'{"Purchase Date":<28}: {equipment["purchase_date"]}') # purchase date
218         print(f'{"Purchase Quantity":<28}: {equipment["purchase_quantity"]}') # purchase quantity
219         print(f'{"Next Scheduled Maintenance":<28}: {equipment["next_scheduled_maintenance"]}') # next schedule maintenance
220         print(f'{"Manufacturer Email":<28}: {equipment["manufacturer_email"]}') # manufacturer email
221         print(f'{"Warranty":<28}: {equipment["warranty"]}') # warranty
222
223         print('')
224         print('-' * 140)
225         print('\nKindly complete the necessary details to submit a report to manager:\n')
```

Figure 4.6.2.5 Function of equipment malfunction report

The function load previous malfunction and maintenance report data from manager notification file to prevent overwrite when saving new report. Next, it calls equipment_list() function to get the selected equipment index, category and other equipment details. As the data is stored in tuple, the function unpacks the tuple and displayed the essential details of selected equipment, including category, equipment name, serial numbers, manufacturer, model number, purchase date, quantity, next scheduled maintenance, manufacturer email, and warranty.

After displaying the equipment details, the function prompts the user to enter the specific serial number of the equipment they want to report. The user input is validated to ensure the serial number exist in selected equipment list. Then, then function ask for the date of malfunction and ensure it is in date format (%d-%m-%Y) and does not greater than current date. The user is also prompted to enter the last maintenance date, which is validated to ensure it is not greater than the malfunction date. Additionally, a detailed description of the incident is requested for further clarification.

The function then confirms whether user want to submit the report. If user agree, a unique and non-duplicate report number of 4 digits is generated. The report is saved to manager

notification file, with information such as report number, category, equipment name, serial number, model number, manufacturer email, warranty, report date, current condition, malfunction date, last maintenance date and description. If user disagree, the report data will not be saved. Last, the user can choose to submit another report or exit the malfunction reporting page. If the user want to submit another report, the function recursively calls itself, else it returns to the equipment management menu by calling equipment_management() function.

Sample output:

```
Please select the equipment you want to report. (enter the number of equipment)
>>> 2

Basic details of selected equipment:

Category : Ovens And Baking Appliances
Equipment Name : Toaster Oven
Serial Number 1 : toasteroven1
Serial Number 2 : toasteroven2
Manufacturer : Nctdream
Model Number : TSTROVN563
Purchase Date : 06-07-2023
Purchase Quantity : 2
Next Scheduled Maintenance : 24-09-2024
Manufacturer Email : nctdream@no1.com
Warranty : 2 years

-----
Kindly complete the necessary details to submit a report to manager:
```

Figure 4.6.2.6 Sample output of selected equipment details

```
-----
Kindly complete the necessary details to submit a report to manager:

1. Date of Report : 30-10-2024
2. Current Condition : Malfunction

Enter the serial number: toasteroven1
Enter the date of malfunction (DD-MM-YYYY): 29-10-2024
Enter the last maintenance date (DD-MM-YYYY): 07-05-2024
Enter a clear description of the malfunction: The toaster oven overheats during use, causing uneven toasting and occasional burning of food.

Confirm submission of malfunction report to manager? (y=yes, n=no)
>>> y

Generating report number...

Malfunction report has been submitted. Thank you for reporting!

Continue reporting? (y=yes, n=no)
>>> y

-----
Welcome to the Equipment Malfunction Report page. Please follow the instructions to report any issues.
```

Figure 4.6.2.7 Sample output of required details of the malfunction report

```

1   {
2     "toasteroven1": {
3       "report_number": "REPORT-3573",
4       "category": "Ovens and Baking Appliances",
5       "equipment_name": "Toaster oven",
6       "serial_number": "toasteroven1",
7       "model_number": "TSTROVN563",
8       "manufacturer_email": "nctdream@no1.com",
9       "warranty": "2 years",
10      "report_date": "30-10-2024",
11      "current_condition": "malfunction",
12      "malfunction_date": "29-10-2024",
13      "last_maintenance_date": "07-05-2024",
14      "description": "The toaster oven overheats during use, causing uneven toasting and occasional burning of food."
15    }
16  }

```

Figure 4.6.2.8 The malfunction report is saved to manager notification file

Figure 4.6.2.6 shows that the function is able to print accurate equipment details based on user selected equipment, while Figure 4.6.2.7 shows that users must enter required details before submitting the report. A unique report number is also generated and avoid duplication. Figure 4.6.2.8 shows that the report submitted is successfully saved to manager notification file.

(b) Maintenance report

```

369 # define function that let user make maintenance report
370 usages += hxlm +1
371 def equipment_maintenance():
372     maintenance_data = load_data_from_manager_notifications() # load previous maintenance report from manager notification file
373
374     print('')
375     print('-' * 140)
376     print('\nWelcome to the Equipment Maintenance Report page. Please follow the instructions to report any issues.')
377     selected_equipment, equipment_info = equipment_lists() # get user selected equipment and its information
378     category, equipment = equipment_info # unpack the tuple to get category and other details
379
380     # display the details of selected equipment
381     max_length = len('Next Scheduled Maintenance')
382
383     print('\nBasic details of selected equipment:\n')
384     print(f'{("Category":<28}: {category.title()}') # category
385     print(f'{("Equipment Name":<28}: {equipment["equipment_name"].title()}') # equipment name
386     index = 1
387     for serial_number in equipment['serial_number']:
388         print(f'{("Serial Number":<28}: {str(index).ljust((max_length + 1) - len("Serial Number"))}: {serial_number})')
389         index += 1
390     print(f'{("Manufacturer":<28}: {equipment["manufacturer"].title()}') # manufacturer
391     print(f'{("Model Number":<28}: {equipment["model_number"]}') # model number
392     print(f'{("Purchase Date":<28}: {equipment["purchase_date"]}') # purchase date
393     print(f'{("Purchase Quantity":<28}: {equipment["purchase_quantity"]}') # purchase quantity
394     print(f'{("Next Scheduled Maintenance":<28}: {equipment["next_scheduled_maintenance"]}') # next schedule maintenance
395     print(f'{("Manufacturer Email":<28}: {equipment["manufacturer_email"]}') # manufacturer email
396     print(f'{("Warranty":<28}: {equipment["warranty"]}') # warranty
397
398     print('')
399     print('-' * 140)
400     print('\nKindly complete the necessary details to submit a report to manager:\n')

```

Figure 4.6.2.9 Function of equipment maintenance needs report

The function is mostly similar as the `equipment_malfunction()` function that submit malfunction report to manager. First, it loads previous malfunction and maintenance report data from manager notification file to prevent overwrite when saving new report. Next, it calls `equipment_list()` function to get the selected equipment index, category and other equipment details. As the data is stored in tuple, the function unpacks the tuple and displayed the essential details of selected equipment, including category, equipment name, serial numbers, manufacturer, model number, purchase date, quantity, next scheduled maintenance, manufacturer email, and warranty.

After displaying the equipment details, the function prompts the user to enter the specific serial number of the equipment they want to report. The user input is validated to ensure the serial number exist in selected equipment list. Then, then function ask for the date of incident (maintenance needed date), ensure it is in date format (%d-%m-%Y) and does not greater than current date. The user is also prompted to enter the last maintenance date, which is validated to ensure it is not greater than the malfunction date. User is required to enter the severity level of issue, which can choose between urgent, high, medium and low, allowing the manager to prioritize task more effectively. Additionally, a detailed description of the incident is requested for further clarification.

The function then confirms whether user want to submit the report. If user agree, a unique and non-duplicate report number of 4 digits is generated. The report is saved to manager notification file, with information such as report number, category, equipment name, serial number, model number, manufacturer email, warranty, report date, current condition, severity, malfunction date, last maintenance date, next scheduled maintenance date and description. If user disagree, the report data will not be saved. Last, the user can choose to submit another report or exit the maintenance reporting page. If the user want to submit another report, the function recursively calls itself, else it returns to the equipment management menu by calling `equipment_management()` function.

Sample output:

```
Please select the equipment you want to report. (enter the number of equipment)
>>> 1

Basic details of selected equipment:

Category : Ovens And Baking Appliances
Equipment Name : Convection Oven
Serial Number 1 : oven1
Manufacturer : Minionnn
Model Number : CVTOVN001
Purchase Date : 03-04-2023
Purchase Quantity : 1
Next Scheduled Maintenance : 15-11-2024
Manufacturer Email : gg@gg.com
Warranty : 1 year

-----
Kindly complete the necessary details to submit a report to manager:
```

Figure 4.6.2.10 Sample output of selected equipment details

```
-----
Kindly complete the necessary details to submit a report to manager:

1. Date of Report : 30-10-2024
2. Current Condition : Maintenance Needed

Enter the serial number: oven1
Enter the date of incident (DD-MM-YYYY): 29-10-2024
Enter the last maintenance date (DD-MM-YYYY): 05-07-2024
Enter the severity of the issue (Choose between: urgent, high, medium, low): urgent
Enter a clear description of the maintenance issue: The convection oven needs maintenance for temperature inconsistency and slow preheating

Confirm submission of maintenance report to manager? (y=yes, n=no)
>>> y

Generating report number...

Maintenance needs report has been submitted. Thank you for reporting!

Continue reporting? (y=yes, n=no)
>>> n

Exit the maintenance report page. Proceeding to equipment management menu.....
```

EQUIPMENT MANAGEMENT MENU

Figure 4.6.2.11 Sample output of required information for maintenance report

```

15 },
16 "oven1": {
17   "report_number": "REPORT-6459",
18   "category": "Ovens and Baking Appliances",
19   "equipment_name": "Convection oven",
20   "serial_number": "oven1",
21   "model_number": "CVTOWN001",
22   "manufacturer_email": "gg@gg.com",
23   "warranty": "1 year",
24   "report_date": "30-10-2024",
25   "current_condition": "maintenance needed",
26   "severity": "urgent",
27   "maintenance_needed_date": "29-10-2024",
28   "last_maintenance_date": "05-07-2024",
29   "next_scheduled_maintenance": "15-11-2024",
30   "description": "The convection oven needs maintenance for temperature inconsistency and slow preheating"
31 }
32 }

```

Figure 4.6.2.12 The maintenance report is saved to manager notification file

Figure 4.6.2.10 shows that the function is able to print accurate equipment details based on user selected equipment, while Figure 4.6.2.11 shows that users must enter required details before submitting the report. A unique report number is also generated and avoid duplication. Figure 4.6.2.12 shows that the report submitted is successfully saved to manager notification file.

4.6.3 Inventory Check

```
# define function that check the availability of ingredients and produce products based on user input quantity
# usages : hxium+1
def recipe_lists():
    while True:
        while True:
            # generate a 4 digit unique report number for each report
            report_num = random.randint( a: 1000, b: 9999 )
            if report_num not in product_list.keys(): # break when the generated report number is not duplicate
                break

            # display the recipe list by category
            print('')
            printed_centered('RECIPE LIST')
            for category, items in recipe_category_groups.items():
                index = 1
                print(f'{category} ')
                for ingredient in items:
                    print(f'{index}. {ingredient.title()}' )
                    index += 1
                print('')

            # prompt user to select a recipe
            recipe_choose = input('What recipe you would like to work on today? Please enter the recipe name. (or enter "cancel" to back to previous page.)\n' 
                                '      >>> ').lower().strip()
            found_recipe = None

            if validation_empty_entries(recipe_choose):
                if recipe_choose.replace( _old: " ", _new: "").isalpha():
                    if recipe_choose == 'cancel': # if user enter cancel, return to baker privilege page
                        return False
                    else:
```

Figure 4.6.3.1 Function to check ingredient availability and carry out production

The function enables checking of ingredient availability and manage production. It loads recipe data, ingredient data and production data from different text files for further processes. It begins by generating a unique, non-duplicate 4-digit random number as the production number. The recipe list is then displayed by category, and the user can select the recipe to work on by entering the name. User input is checked and validated to ensure that correct and valid recipe name is entered.

If valid recipe is chosen, the function will display the selected recipe details. After that, user is prompt to enter the quantity they wish to produce. The function will check whether the current ingredients in inventory are available for the production quantity, by multiplying the ingredient needed for one unit of product with production quantity entered by user. Unit conversions are also handled if necessary (for example, kg to g, g to kg, l to ml, ml to l and more) to ensure accurate calculations.

If the quantity needed for production is smaller than current quantity of ingredient in inventory, the ingredient is considered available. If all ingredients are available, user is prompt whether to confirm production. If user confirm, the quantity of ingredients in inventory will be subtracted by the quantity used, and production detail is saved to inventory check file with the

unique report number as the key. If at least one ingredient is unavailable, a message will display the unavailable ingredient, and the maximum possible quantity that can be produced with current stock. Last, the user is prompted to continue with another recipe or exit. If the user wants to continue, the function recursively calls itself, else it returns to baker privilege page.

Sample output:

```
-----  
          RECIPE LIST  
-----  
  ♡ Cakes ♡  
1. Tiramisu  
2. Matcha Layer Cake  
3. Red Velvet  
  
  ♡ Biscuits ♡  
1. Bello  
  
  ♡ Pastries ♡  
1. Applepie  
  
What recipe you would like to work on today? Please enter the recipe name. (or enter "cancel" to back to previous page.)  
>>> matcha layer cake  
  
Here are the details for your chosen recipe:
```

Figure 4.6.3.2 Sample output of selecting recipe to work on

```
Here are the details for your chosen recipe:
```

```
Recipe Category      : Cakes
Recipe Name         : matcha layer cake
```

```
Ingredient Used:
```

```
1. Baking Powder x 300.0 g
2. Almond Flour   x 40.0 g
3. Maple Syrup    x 25.0 ml
```

```
Ingredient Notes   : layer cake yuds
```

```
Equipment Used:
```

```
1. Stand Mixer
2. Convection Oven
```

```
Baking Temperature (°C) : 90
```

```
Baking Time (min)      : 50
```

```
Instructions:
```

```
1. Mix The Baking Powder With Almond Flour
2. Bake It
3. Add Almond Flour, Whipping Cream And Maple Syrup On The Baked Cake
```

```
Please enter the quantity you want to produce: 10
```

```
Checking the availability of required ingredient.....
```

```
Not enough baking powder in stock. You can produce a maximum of 6.67 units of matcha layer cake.
```

```
Please enter the quantity you want to produce:
```

Figure 4.6.3.3 Sample output of ingredient unavailable scenario

```
Please enter the quantity you want to produce: 3
```

```
Checking the availability of required ingredient.....
```

```
All ingredients needed available!
```

```
Are you sure you want to produce 3 units of matcha layer cake? Enter y for yes or n for no: y
```

```
Successfully added!
```

```
Continue working on another recipe? (y=yes, n=no)
```

```
>>> y
```

```
-----  
          RECIPE LIST  
-----
```

```
  ↗ Cakes ↗
```

```
1. Tiramisu
```

Figure 4.6.3.4 Sample output of ingredient available scenario

```

19     },
20     "3312": {
21       "recipe_category": "Cakes",
22       "recipe_name": "matcha layer cake",
23       "production_quantity": 3,
24       "date_of_production": "31-10-2024"
25     }
26   }

```

Figure 4.6.3.5 The production detail is added to the inventory check file

Figure 4.6.3.2 shows how the function displays the recipe list and let user to select desired recipe. User's input will be validated to ensure it exists in given list. If user enter a valid recipe, the function will display the recipe details and prompt user to enter production quantity. The function will then check whether the ingredient needed is enough for the production quantity. Figure 4.6.3.3 shows the scenario when the ingredient is unavailable, where the function will print the unavailable ingredient name, and the maximum quantity that can be produced by current stock. Figure 4.6.3.4 shows the scenario that all ingredients is available, where user can select to confirm the production. If yes, Figure 4.6.3.5 shows the production details is successfully added to baker inventory check file. If no, the production will be cancelled. User is then prompted to select whether to continue work on another recipe. If the user wants to continue, the function recursively calls itself, else it returns to baker privilege page.

4.6.4 Product Keeping

```
113 # define function to display product management option
114 # usages : hxium+1
115 def product_management():
116     print('\n-----')
117     print('\t\t\t\t', 'PRODUCT MANAGEMENT')
118     print('-----')
119     while True:
120         # initialize the number of unsaved product to 0
121         unsaved_product = 0
122
123         # if there is any unsaved product
124         for product in unsaved_product_list.values():
125             unsaved_product += 1 # add the number of unsaved product
126
127         # display how many unsaved product
128         if unsaved_product != 0:
129             print(
130                 f"\n⚠️ Notification: {unsaved_product} product(s) pending for record.")
131             break
132         else:
133             pass
134
135         print('\n1. Add Product')
136         print('2. Update Product')
137         print('3. Remove Product')
138         print('4. Back to Previous Page')
139         print('')
140
141         while True:
142             # collect user selected product management option
143             option_product_management = input('Please choose a service:'
```

Figure 4.6.4.1 Function of product management options

The function starts by reading each unsaved product entry from the baker inventory check file and increments the unsaved_product variable (initially set to 0) by 1 for each unsaved item found. Next, it displays the title, ‘PRODUCT MANAGEMENT’ and total number of unsaved products found. It also displays a list of product management options, including add product, update product, remove product and back to previous page. It then enters a loop, prompting user to select an option, the loop will only break until a valid option is found. If user input ‘1’, the function will call product_details() function to add a product, if user input ‘2’, it will call update_product() function to update product, if user input ‘3’, it will call delete_product() function to delete a product and if user input ‘4’ it will return False and break the loop.

Sample Output:

```
-----  
          PRODUCT MANAGEMENT  
-----  
  
⚠ Notification: 3 product(s) pending for record.  
  
1. Add Product  
2. Update Product  
3. Remove Product  
4. Back to Previous Page  
  
Please choose a service:  
>>> 5  
  
+-----+  
|⚠ Please enter a valid number.|  
+-----+  
  
Please choose a service:  
>>> d  
  
+-----+  
|⚠ Please enter a valid number.|  
+-----+  
  
Please choose a service:  
>>>
```

Figure 4.6.4.2 Sample output of product management options

The output shows that user is able to make clear and informed decisions by viewing the instructions, and the system also ensure that the input is valid before heading to next step. This validation helps prevent errors and provides a smooth and guided experiences.

(b)(a) Add product

```
# define the function to let user select which unsaved product to record and record its category
usage ▲ hxdum +1
def product_categories():
    # loop through and display unsaved product and prompt user to select one for recording
    while True:
        index = 1
        print('')
        printed_centered('UNSAVED PRODUCT LIST')
        for key, value in unsaved_product_list.items():
            print(f'{index}.')
            for title, details in value.items():
                print(f'{title.replace("_", " "):<20}: {details}')
            index += 1

        try:
            index_of_product_to_edit = int(
                input(
                    f'\nEnter index number to record the product to system (or enter {len(unsaved_product_list)} + 1} to cancel):\n>>> ')
            if index_of_product_to_edit == len(unsaved_product_list) + 1: # if "Cancel" is selected
                print('\nCancelling. Exiting to Product Management page.....') # return to the previous page
                product_management()
                break

            elif 1 <= index_of_product_to_edit <= len(unsaved_product_list): # if valid index is selected

                while True:
                    selected_unsaved_product = list(unsaved_product_list.keys())[index_of_product_to_edit - 1] # append all the keys into a list and identify the selected product by indexing
                    print('')
                    print(
                        f'! You are now recording {unsaved_product_list[selected_unsaved_product]["recipe_name"].title()}\\'s data.')
                    break
        except ValueError:
            print('Please enter a valid integer value between 1 and', len(unsaved_product_list))
            continue
```

Figure 4.6.4.3 Function for user to enter product category and name

The function begins by displaying the basic details of unsaved products retrieved from baker inventory check file for user to select a product to record. Each unsaved product is assigned a numbered index for user to select. The function will validate if user's entered index number is fall within the valid range, if no, user is prompted to enter again. User can also choose to cancel the process by selecting the last index number.

If a valid index number is entered, the selected unsaved product is identified by its position (the given index number) and the function will prompt user to start record details for the chosen product. User is prompted to enter the category (Breads, Cakes, Pastries, Biscuits, Muffins, and Others) of the product, which is validated to not only matches the given category list, but also matches the predefined category of the selected unsaved product, otherwise the user is prompt to choose again. If the category aligns, the function will return the selected category and selected index of unsaved product.

Sample Output:

```
-----  
          UNSAVED PRODUCT LIST  
-----  
  
1.  
recipe category      : Biscuits  
recipe name          : minion biscuit  
production quantity : 2  
date of production  : 09-04-2024  
  
2.  
recipe category      : Biscuits  
recipe name          : santa gingerbread  
production quantity : 40  
date of production  : 07-07-2024  
  
3.  
recipe category      : Biscuits  
recipe name          : choco cookies  
production quantity : 30  
date of production  : 30-10-2024  
  
Enter index number to record the product to system (or enter 4 to cancel):  
>>> 1  
  
! You are now recording Minion Biscuit's data.
```

Figure 4.6.4.3 Sample output for user to choose an unsaved product

```
1  {  
2    "7368": {  
3      "recipe_category": "Biscuits",  
4      "recipe_name": "minion biscuit",  
5      "production_quantity": 2,  
6      "date_of_production": "09-04-2024"  
7    },  
8    "2792": {  
9      "recipe_category": "Biscuits",  
10     "recipe_name": "santa gingerbread",  
11     "production_quantity": 40,  
12     "date_of_production": "07-07-2024"  
13   },  
14   "4970": {  
15     "recipe_category": "Biscuits",  
16     "recipe_name": "choco cookies",  
17     "production_quantity": 30,  
18     "date_of_production": "30-10-2024"  
19   }  
20 }
```

Figure 4.6.4.4 Unsaved product data saved in baker inventory check file

```
| You are now recording Minion Biscuit's data.

-----
MAIN TYPES OF BAKERY PRODUCTS
-----
1. Breads
2. Cakes
3. Pastries
4. Biscuits
5. Muffins
6. Others
7. Back to Previous Page

Please choose the name of category by enter index number:
>>> 2

+-----+
|⚠ Category must same as recipe category. |
+-----+

Please choose the name of category by enter index number:
>>> 4

+-----+
|💡 Fill out the following fields to add a new product to the inventory. |
+-----+
1. Quantity Produced      :
```

Figure 4.6.4.5 Function for user to enter product category and name

Figure 4.6.4.3 shows that the function will display all unsaved products details that retrieve from baker inventory check file for user to choose. Figure 4.6.4.4 shows the unsaved product data stored in baker inventory check file. The function validates whether the entered index number falls within the valid range; if it does not, the user is prompted to enter a valid number again. Figure 4.6.4.5 shows that the function will validate the input category to ensure it matches both value in category list and the predefined category of unsaved product before proceeding to next step.

```

258 # define function to record the product details
259 # usages : hxium +1
260 def product_details():
261     # get category and index of selected product
262     category, index_of_product_to_edit = product_categories()
263     # get the selected product using the index
264     selected_unsaved_product = list(unsaved_product_list.keys())[index_of_product_to_edit - 1]
265
266     # define the title required for product details
267     product_info = ['Product Name', 'Product Code', 'Batch Number', 'Date of Production', 'Shelf Life (_ days)',
268                     'Expiry Date (DD-MM-YYYY)', 'Recipe', 'Quantity Produced', 'Baker\'s Username', 'Allergens']
269
270     # calculate the maximum length of title for formatting
271     max_length = 0
272     for item in product_info:
273         if len(item) > max_length:
274             max_length = len(item)
275
276     print('\n+-----+')
277     print('! Fill out the following fields to add a new product to the inventory. !')
278     print('+-----+')
279
280     while True:
281         # prompt user to enter quantity produced and ensure it matches the production quantity of the selected unsaved product
282         quantity_produced = input(f'1. {product_info[7].ljust(max_length + 2)}: ').strip()
283         if validation_empty_entries(quantity_produced):
284             if quantity_produced.isdigit():
285                 if int(quantity_produced) == int(unsaved_product_list[selected_unsaved_product]["production_quantity"]):
286                     quantity = int(quantity_produced)
287                     break
288                 else:
289                     print('\n+-----+')

```

Figure 4.6.4.6 Function for user to enter required details for product record keeping

This function will guide user through the process of adding details for an unsaved product to the inventory. First, it gets the return value (selected category, index of selected unsaved product) from product_categories() function.

For each required product details, the function will validate user input. First, the function will ask user to enter the quantity produced, and the input is ensured to match the predefined production quantity of selected unsaved product. Next, the product name must only contain alphabet character, while the product code and batch number must in alphanumeric format and does not duplicate with other product record-keeping data in baker record keeping file.

The date of production is validated to matches the predefined production date of selected unsaved product. The shelf life must fall within the limit based on the product category, where maximum 5 days for breads and muffins, maximum 7 days for cakes and pastries and maximum 14 days for biscuits and others. The expiry date is also validated to be within the date of production added by shelf life plus one day. For the baker's name, the input is validated so that it can be matched with the baker username in baker file. The allergens can contain only alphabet and underscores (act as spacing). Last, the function will prompt user to enter serial number,

and the frequency is based on the quantity produced. The serial number is validating so that it is unique and does not duplicate within the baker product keeping file.

Finally, the function saves the product details recorded by user into product keeping file and removed the selected unsaved product from baker inventory check file. The function will also ask user either to continue adding another product or return to the product management page.

Sample output:

```
+-----+
| ⚡ Fill out the following fields to add a new product to the inventory. |
+-----+
1. Quantity Produced      : 2
2. Product Name           : Minion Biscuit
3. Product Code            : MIN030
4. Batch Number            : B001
5. Date of Production     : 09-04-2024
6. Shelf Life (__ days)   : 8
6. Expiry Date (DD-MM-YYYY) : 11-11-2024

+-----+
| ▲ The expired date does not fall within the allowable period.
|   The allowable period must between the date of production and date of production + shelf life + 1 day.
| * Allowable period: 09-04-2024 to 18-04-2024.
+-----+

6. Expiry Date (DD-MM-YYYY) : 17-04-2024
8. Baker's Username        : christine

+-----+
| ⚡ If there is more than one data item, separate them with a space.
| ⚡ If a name consists of more than one words, use underscore (_) to represent the space, e.g. tree_nuts.
+-----+

9. Allergens                : gluten egg milk
Enter serial number for item 1: biscuit1
Enter serial number for item 2: biscuit2
```

Figure 4.6.4.7 Sample output of required details needed for product record-keeping

```

189     "B001": {
190         "category": "Biscuits",
191         "product_name": "minion biscuit",
192         "product_code": "MIN030",
193         "serial_number": [
194             "biscuit1",
195             "biscuit2"
196         ],
197         "quantity_produced": "2",
198         "batch_number": "B001",
199         "date_of_production": "09-04-2024",
200         "shelf_life": "8",
201         "expiry_date": "17-04-2024",
202         "baker_name": "christine",
203         "allergens": [
204             "gluten",
205             "egg",
206             "milk"
207         ]
208     }

```

Figure 4.6.4.8 Sample output of product details saved to baker product keeping file

```

1   {
2       "2792": {
3           "recipe_category": "Biscuits",
4           "recipe_name": "santa gingerbread",
5           "production_quantity": 40,
6           "date_of_production": "07-07-2024"
7       },
8       "4970": {
9           "recipe_category": "Biscuits",
10          "recipe_name": "choco cookies",
11          "production_quantity": 30,
12          "date_of_production": "30-10-2024"
13      }
14  }

```

Figure 4.6.4.9 Sample output of remaining unsaved product data

Figure 4.6.4.7 shows the information required for product record-keeping. The function will also accurately validate the input of user to ensure correct data is saved. Figure 4.6.4.8 shows

the product data entered by user is successfully saved into baker product keeping file. Figure 4.6.4.9 shows the remaining unsaved product data after the selected product is being recorded, proof that the selected unsaved product has been deleted after the required information is entered, validated and saved.

(e)(b) Update product

```

536 # define function to update product information
537 # usages  hxium+1
538 def update_product():
539     while True:
540         index = 1
541         print('')
542         printed_centered('PRODUCT LIST')
543         for batch_number, product in product_data.items():
544             print(f'{index}. {product["product_name"].title()}') # display product name with index number
545             index += 1
546         print(f'{len(product_data) + 1}. Cancel')
547
548         # get user selected product to update
549         try:
550             index_of_product_to_edit = int(input(
551                 f'\nEnter index number to update the information of the product (or enter {len(product_data) + 1} to cancel):\n>>> '))
552
553             if index_of_product_to_edit == len(product_data) + 1: # if "Cancel" is selected
554                 print('\n Cancelling, Exiting to Product Management page.....') # return to the previous page
555                 product_management()
556                 break
557
558             # ensure the selected index is within valid range
559             elif 1 <= index_of_product_to_edit <= len(product_data):
560                 for batch_number, product in product_data.items():
561                     # append all the keys into a list and identify the selected product by indexing
562                     selected_product_key = list(product_data.keys())[index_of_product_to_edit - 1]
563                     break
564
565             else:
566                 print('\n ! Product not found.') # error displayed if selected product not found
567                 continue

```

Figure 4.6.4.10 Function for update product data

The function begins by displaying the product stored in baker product keeping file and assign them a numbered index for user to choose. The user input index is validated so that it falls within the valid range. User can also choose the last index number to cancel the update process. If the selected index is valid, the function will prompt user to enter the specific attribute they want to update. It will then ensure the selected attribute exists in the product details. If the selected attribute exists, the function will prompt user to enter a new value and validate it. User cannot update the data for production quantity, production date and category.

When modifying serial number, the function displays the existing serial number and prompt user to select a serial number. After validating the serial number exists within the selected product, user can enter a new alphanumeric value to replace the original. The function will

replace the old serial number with new value and save the data after validating the serial number does not duplicate with other serial number in baker product keeping file.

When modifying allergen attribute, the function provides users 3 options: to add, edit or delete allergens. The function will validate user selected option to ensure is between these options and carry out corresponding process. If user choose to add, they can input new allergens until they type ‘done’ to finish the process. If user choose to edit, it will display current allergens with index number, and user can select an allergen by index and input a replacement allergen. If user choose to delete, it will display current allergens with index number also, and user can select an allergen by index, and the function will remove it from the allergen list. The data updated will be saved after validation to ensure all inputs are alphabetic.

For general attributes (other than allergens and serial number), the function will prompt user to enter a new value and validate the input. The validation requirement for each attribute is same as validation in product_details() function. If user enter an invalid input, it will prompt user to enter again. If the new value is valid, it will update the old value of selected attribute to new value and save the data.

Sample Output:

```
-----  
          PRODUCT LIST  
-----  
1. Banana Bread  
2. Apple Pie  
3. Cheezy Bun  
4. Tiramisu  
5. Apple Cake  
6. Bello Minion Biscuit  
7. Vanilla Muffin  
8. Garlic Bun  
9. Matcha Layer Cake  
10. Minion Biscuit  
11. Cancel  
  
Enter index number to update the information of the product (or enter 11 to cancel):  
>>> 8  
-----  
          GARLIC BUN'S DATA  
-----
```

Figure 4.6.4.11 Sample output of product list for user to select and update

The figure shows that the function retrieve product data from baker product keeping file and display them with corresponding index number. The function also validates the user input, ensure it is valid before proceeding to next step.

```
-----  
          GARLIC BUN'S DATA  
-----  
  
category: Breads  
product_name: garlic bun  
product_code: GR013  
serial_number: ['bun1', 'bun2', 'bun3']  
quantity_produced: 3  
batch_number: 011  
date_of_production: 01-11-2024  
shelf_life: 5  
expiry_date: 06-11-2024  
baker_name: christine  
allergens: ['garlic']  
  
To change the information, please enter the exact matching name. Example: product_name.  
* Note: category, date of production, and quantity produced cannot be changed.  
  
Which information do you want to update? (or enter "cancel")  
>>> category  
  
! Data not found.
```

Figure 4.6.4.12 Sample output of product details of selected product

The figure shows that after ensuring the input index is valid, the function will display the product details of selected product correctly. Next, it will prompt user to input an attribute to update. If user input does not find in the given product data, or user enter attributes such as quantity produce, category and date of production which cannot be changed, it will display error message and prompt user to enter again.

```

shelf_life: 5
expiry_date: 06-11-2024
baker_name: christine
allergens: ['garlic']

To change the information, please enter the exact matching name. Example: product_name.
* Note: category, date of production, and quantity produced cannot be changed.

Which information do you want to update? (or enter "cancel")
>>> serial_number

Serial number 1 : bun1
Serial number 2 : bun2
Serial number 3 : bun3

Please enter the serial number you want to edit (or enter 'cancel')
>>> bun1

Enter the new serial number: bun75635

Information saved.

```

Figure 4.6.4.12 Sample output when selected attributes is serial number

The figure shows that the function will display the existing serial number and prompt user to enter the serial number they want to update. After validating the serial number exists within the selected product, user can enter a new alphanumeric value to replace the original. The function will replace the old serial number with new value and save the data after validating the serial number does not duplicate with other serial number in baker product keeping file.

```

To change the information, please enter the exact matching name. Example: product_name.
* Note: category, date of production, and quantity produced cannot be changed.

Which information do you want to update? (or enter "cancel")
>>> allergens

1. garlic

Add, edit or delete allergens?
Please enter 'add', 'edit' or 'delete' (or enter 'cancel' to stop): add

Enter the new allergen (enter 'done' to stop): gluten

Enter the new allergen (enter 'done' to stop): oat

Enter the new allergen (enter 'done' to stop): done

Information saved.

```

Figure 4.6.4.13 Sample output when selected attributes is allergens

The figure shows the scenario when user choose to add new allergens to current allergen list. The function will display the existing allergens and prompt user to enter new allergens until they type ‘done’ to finish the process. The data updated will be saved after validation to ensure all inputs are alphabetic.

```
To change the information, please enter the exact matching name. Example: product_name.  
* Note: category, date of production, and quantity produced cannot be changed.  
  
Which information do you want to update? (or enter "cancel")  
>>> expiry_date  
  
Enter new expiry_date: 03-11-2024  
  
expiry_date of 011 is updated.
```

Figure 4.6.4.14 Sample output when selected attributes is general attribute

The figure shows the scenario when user select general attribute (other than allergens and serial number). The function will prompt user to enter a new value and validate the input. If the new value is valid, it will update the old value of selected attribute to new value and save the data.

```
-----  
GARLIC BUN'S DATA  
-----  
  
category: Breads  
product_name: garlic bun  
product_code: GR013  
serial_number: ['bun75635', 'bun2', 'bun3']  
quantity_produced: 3  
batch_number: 011  
date_of_production: 01-11-2024  
shelf_life: 5  
expiry_date: 03-11-2024  
baker_name: christine  
allergens: ['garlic', 'gluten', 'oat']  
  
To change the information, please enter the exact matching name. Example: product_name.  
* Note: category, date of production, and quantity produced cannot be changed.  
  
Which information do you want to update? (or enter "cancel")  
>>>
```

Figure 4.6.4.15 Sample output of updated product data when displayed

```
155     "011": {
156         "category": "Breads",
157         "product_name": "garlic bun",
158         "product_code": "GR013",
159         "serial_number": [
160             "bun75635",
161             "bun2",
162             "bun3"
163         ],
164         "quantity_produced": "3",
165         "batch_number": "011",
166         "date_of_production": "01-11-2024",
167         "shelf_life": "5",
168         "expiry_date": "03-11-2024",
169         "baker_name": "christine",
170         "allergens": [
171             "garlic",
172             "gluten",
173             "oat"
174         ]
175     },
```

Figure 4.6.4.16 Sample output of updated product data in text file

Figure 4.6.4.15 and Figure 4.6.4.16 shows that the product data is successfully updated and saved in baker product keeping file.

(d)(c) Delete product

```
def delete_product():
    while True:
        # display product list with index number
        index = 1
        print('\n-----')
        print('t\ t\ t\ t', ', ' , 'PRODUCT LIST')
        print('-----')
        for name, info in product_data.items():
            print(f'{index}. {name.title()}' )
            index += 1
        print(f'{len(product_data) + 1}, cancel') # option to cancel the process

        try:
            # prompt user to select a product by its index
            index_remove_product = int(
                input(f'\nWhich product do you want to remove? (or enter {len(product_data) + 1} to cancel)\n>> '))
            if index_remove_product == len(product_data) + 1: # cancel the process
                print('\nCancelling. Exiting to Services page.....')
                product_management()
                break

            # if selected index falls within valid range
            elif 1 <= index_remove_product <= len(product_data):
                product_to_remove = list(product_data.keys())[index_remove_product - 1] # identify product to remove by accessing the index of key of product
                del product_data[product_to_remove] # delete the selected product
                save_info(product_data)
                print(
                    f'\n{product_to_remove.title()} is removed.\n') # inform user that the selected product is removed successfully

        while True:
```

Figure 4.6.4.17 Function for delete product data

The function begins by displaying the product stored in baker product keeping file and assign them a numbered index for user to choose. The user input index is validated so that it falls within the valid range. User can also choose the last index number to cancel the update process. If user choose the cancel option, they will return to product management page. If the selected index is valid, the function identifies the corresponding product by converting the index to the product name key. Next, it removes the product data based on the product name key and save the data.

After deletion, the function prompt user whether to continue delete another product. If user choose yes, the deletion process continues from the beginning of function. If user choose no, it will exit the loop and call product_management() function to return to product management menu. If the input is invalid it will prompt user to enter again.

Sample Output:

```
-----  
          PRODUCT LIST  
-----  
1. 001 - Banana Bread  
2. 003 - Apple Pie  
3. 004 - Cheezy Bun  
4. 009 - Tiramisu  
5. 007 - Apple Cake  
6. 008 - Vanilla Muffin  
7. 011 - Garlic Bun  
8. 012 - Matcha Layer Cake  
9. B001 - Minion Biscuit  
10. cancel  
  
Which product do you want to remove? (or enter 10 to cancel)  
>>> 1  
  
001 is removed.  
  
Continue to remove? (y=yes, n=no)  
>>> n  
  
Stop removing. Exiting to Services page.....  
  
-----  
          PRODUCT MANAGEMENT  
-----  
  
⚠ Notification: 2 product(s) pending for record.
```

Figure 4.6.4.18 Sample output of product delete process

The figure shows that the function retrieve product data from baker product keeping file and display them with corresponding index number. After ensuring the input is valid, it will remove the data of selected product and display a confirmation message. Next, it will prompt user whether to continue delete another product and carry out corresponding process based on user's input.

4.6.5 Recipe Management

```
5 usages ± hxlum
135 def recipe_management():
136     printed_centered('RECIPE MANAGEMENT')
137     print('\n1. Add Recipe')
138     print('2. Update Recipe')
139     print('3. Remove Recipe')
140     print('4. Back to Previous Page')
141
142     # loop until a valid choice is selected or user choose to exit
143     while True:
144         option_recipe_management = input('\nPlease choose a service by entering its index number:'
145                                         '\n>>> ')
146         if validation_empty_entries(option_recipe_management):
147             if option_recipe_management not in ['1', '2', '3', '4']:
148                 print('\n+-----+')
149                 print('| ▲ Please enter a valid index number. |')
150                 print('+-----+')
151             else:
152                 if option_recipe_management == '1':
153                     recipe_instruction()
154                     break
155                 elif option_recipe_management == '2':
156                     update_recipe()
157                     break
158                 elif option_recipe_management == '3':
159                     delete_recipe()
160                     break
161                 elif option_recipe_management == '4':
162                     print('\nExiting to the previous page.....')
163                     break
164
```

Figure 4.6.5.1 Function of recipe management menu

The function displays the title, ‘RECIPE MANAGEMENT’ and a list of recipe management options, including add, update, remove and back to previous page. It then enters a loop, prompting user to select an option, the loop will only break until a valid option is found. If user input ‘1’, the function will call `recipe_instruction()` function to add a recipe, if user input ‘2’, it will call `update_recipe()` function to update recipe, if user input ‘3’, it will call `delete_recipe()` function to delete a recipe and if user input ‘4’ it will return False and break the loop.

Sample Output:

```
-----  
          RECIPE MANAGEMENT  
-----  
  
1. Add Recipe  
2. Update Recipe  
3. Remove Recipe  
4. Back to Previous Page  
  
Please choose a service by entering its index number:  
>>> 5  
  
+-----+  
| ▲ Please enter a valid index number. |  
+-----+  
  
Please choose a service by entering its index number:  
>>> d  
  
+-----+  
| ▲ Please enter a valid index number. |  
+-----+  
  
Please choose a service by entering its index number:  
>>>
```

Figure 4.6.5.2 Sample output of recipe management menu

The output shows that user is able to make clear and informed decisions by viewing the instructions, and the system also ensure that the input is valid before heading to next step. This validation helps prevent errors and provides a smooth and guided experiences.

(a) Add recipe

```
166 # define function that let user enter recipe category and recipe name
167 usage += hulum+1
168 def create_recipe():
169     recipe_info = ['Recipe Name', 'Recipe\'s Category'] # list of required information for the recipe
170
171     # calculate the maximum length of items in recipe_info for formatting purpose
172     max_length = 0
173     for item in recipe_info:
174         if len(item) > max_length:
175             max_length = len(item)
176
177     # prompt user to enter recipe category and ensure the input fall within given categories
178     while True:
179         print('\n💡 Categories: Breads, Cakes, Pastries, Biscuits, Muffins, Others 💡')
180         category = input(f'1. {recipe_info[1].ljust(max_length + 2)}: ')
181         if validation_empty_entries(category):
182             if category.strip().isalpha():
183                 if category in ['Breads', 'Cakes', 'Pastries', 'Biscuits', 'Muffins', 'Others']:
184                     break
185                 else:
186                     print('-----+')
187                     print('⚠ Please enter a valid category based on the categories given. (Case sensitive.)')
188                     print('-----+')
189             else:
190                 print(
191                     '\n-----+')
192                 print(
193                     '|⚠ Please enter a valid category. (Cannot contain any spacing, digits and special characters.)')
194                 print(
195                     '-----+')
```

Figure 4.6.5.3 Function that prompt user to enter recipe category and name

The add recipe feature is divided to four parts. This is the first part, where functions to prompt user to enter the recipe category and recipe name. If user input category does not exist in given category list, user is prompt to enter again. For the recipe name, user input is validated to ensure only contains alphabet characters and does not duplicate with another recipe name in baker recipe file. If all inputs are valid, the function will return the value and proceed to next step.

Sample Output:

```
-----  
RECIPE MANAGEMENT  
-----  
  
1. Add Recipe  
2. Update Recipe  
3. Remove Recipe  
4. Back to Previous Page  
  
Please choose a service by entering its index number:  
>>> 1  
  
💡 Categories: Breads, Cakes, Pastries, Biscuits, Muffins, Others 💡  
1. Recipe's Category : Cakes  
2. Recipe Name : cheese cake
```

Figure 4.6.5.4 Sample output of recipe category and name

The output shows how the function prompt and display required details for user to input. Additionally, it shows that user input is accurately validated before returning the value.

```
224 # define the function that let user add ingredient for recipe  
225 # usages : hxium+1  
226 def recipe_ingredient(adding_ingredient, update_ingredient):  
227     ingredients = [] # initialize an empty list to store selected ingredients  
228     add_notes = True # create a flag to help in exit loop  
229     ingredient_notes = None # initialize ingredient_notes to None  
230  
231     while add_notes:  
232         # loop until valid ingredient name is selected  
233         while True:  
234             printed_centered('INGREDIENT LIST')  
235             for category, items in ingredient_category_groups.items():  
236                 index = 1  
237                 print(f' {category} ')  
238                 for ingredient in items:  
239                     print(f'{index}. {ingredient.title()}') # print the index and ingredient name  
240                     index += 1  
241             print()  
242  
243             # prompt user to select an ingredient  
244             ingredient_name = input(f' 🎨 Enter the ingredient name: ').strip()  
245             found_category = None  
246  
247             # ensure the input ingredient is existed in the list given  
248             if validation_empty_entries(ingredient_name):  
249                 if ingredient_name.replace(" ", "").isalpha():  
250                     if is_ingredient_duplicate(ingredient_name, ingredients): # ensure the entered ingredient does not duplicate  
251                         for category, items in ingredient_category_groups.items():  
252                             for item in items:  
253                                 if ingredient_name.lower() == item.lower():  
254                                     found_category = category # track the category of selected ingredient
```

Figure 4.6.5.5 Function that prompt user to enter ingredient

The function prompt user to add ingredients to a recipe. First, it initialized an empty list named ingredients to store the selected ingredients. The function load data from inventory ingredient file and display categorized ingredient list, and let the user selects an ingredient by name. User input is validated to ensure it exists in ingredient list.

Next, the function prompt user to enter a valid unit measurement based on the ingredient category and comparing the input with predefined allowable unit. (for example, g and kg for flours and grains) Once valid input is entered, the function prompts for quantity per unit and checking that it is greater than 0. If all input is valid and non-duplicate with previous input ingredients, the ingredient name, unit measurement and quantity are stored in the ingredients list and display it.

If the adding_ingredient flag is True, the function prompt user whether to add more ingredients. If user choose to add more, the function will loop again. If user choose not to add more, and if update_ingredient is True, the user can add optional notes for the ingredient list. The function then exits the loop and return the ingredients list and ingredient note.

Sample Output:

```
-----  
          INGREDIENT LIST  
-----  
* Flours and Grains *  
1. Whole Wheat Flour  
2. Cornmeal  
3. Almond Flour  
  
* Sweeteners *  
1. Maple Syrup  
2. Candy  
  
* Dairy and Non-Dairy Products *  
1. Whipping Cream  
  
* Fruits and Vegetables *  
1. Apple  
  
* Spices and Flavourings *  
1. Chilly  
  
* Leavening Agents *  
1. Baking Powder  
  
✍ Enter the ingredient name: whole wheat flour
```

Figure 4.6.5.6 Sample output of ingredient list

```

    ↗ Enter the ingredient name: whole wheat flour
    ↗ Allowable unit measurement: g, kg
    ↗ Enter the unit measurement of whole wheat flour: g
    ↗ Enter the quantity per unit of whole wheat flour: 400

Ingredient so far:
1. Whole Wheat Flour x 400.0 g

Continue adding ingredients? (y=yes, n=no)
>>> y
-----
          INGREDIENT LIST
-----
    ↗ Flours and Grains ↗
1. Whole Wheat Flour
2. Cornmeal
3. Almond Flour

```

Figure 4.6.5.7 Sample output if user choose to enter another ingredient

```

    ↗ Enter the ingredient name: maple syrup
    ↗ Allowable unit measurement: g, ml
    ↗ Enter the unit measurement of maple syrup: ml
    ↗ Enter the quantity per unit of maple syrup: 20

Ingredient so far:
1. Whole Wheat Flour x 400.0 g
2. Maple Syrup      x 20.0   ml

Continue adding ingredients? (y=yes, n=no)
>>> n

Any additional details or notes you'd like to include for these ingredients? If not, enter 'no'.
>>> maple syrup must store in fridge

Note added: maple syrup must store in fridge

Stop adding. Proceeding to select the necessary equipment 😊

```

Figure 4.6.5.8 Sample output if user choose not to enter another ingredient

Figure 4.6.5.6 shows the function display the ingredient list and prompt user to enter selected ingredient. Figure 4.6.5.7 shows Figure 4.6.5.7 shows the function validate and prompt user to

enter ingredient name, quantity and unit measurement. If user choose to continue adding ingredient, the function will loop again. Figure 4.6.5.8 shows the scenario when user choose to not continue adding ingredient. The function will prompt user to enter ingredient note, exit the loop and return value.

```

409     def recipe_equipment(update_equipment):
410         equipments = [] # initialize an empty list to store selected equipment
411         printed_centered('EQUIPMENT LIST')
412         for category, items in equipment_category_groups.items():
413             index = 1
414             print(f'\t {category}\t')
415             for equipment in items:
416                 print(f'{index}. {equipment.title()}') # print the index with equipment name
417                 index += 1
418             print('')
419             print('💡 Please enter the name of selected equipment (or type "done" to finish)')
420
421             # if update_equipment equals to True, prompt user to enter equipment
422             if update_equipment:
423                 while True:
424                     equipment_name = input(f'\n✍ Enter the name of selected equipment {len(equipments) + 1}: ').lower().strip()
425
426                     # if user choose to stop enter
427                     if equipment_name == 'done':
428                         if len(equipments) == 0: # ensure user have entered at least one equipment
429                             print('\n+-----+')
430                             print('⚠ You must enter at least one equipment before finishing. !')
431                             print('+-----+')
432                             continue
433                     else:
434                         print('\nStop adding. Continue to Recipe Instruction page.....')
435                         break
436
437             # ensure the selected equipment is existed in the list given

```

Figure 4.6.5.9 Function that prompt user to enter equipment

The function prompt user to add equipment to a recipe. First, it initializes an empty list named equipments to store the selected equipment. The function load data from baker equipment file and displays a categorized equipment list. If add_equipment is True, it allows user to select equipment by name. User input is validated to ensure it exists in the displayed equipment list before being appended to the equipments list. User can enter ‘done’ to stop adding equipment. The function then checks whether equipments list contains any items, if not, it displays a message, prompt user to add at least one equipment. If equipments list have values, it will exit the loop and return the list.

Sample Output:

```
-----  
          EQUIPMENT LIST  
-----  
  ↗ Ovens and Baking Appliances ↗  
1. Convection Oven  
2. Toaster Oven  
  
  ↗ Mixing Equipment ↗  
1. Stand Mixer  
  
💡 Please enter the name of selected equipment (or type "done" to finish)  
  
📝 Enter the name of selected equipment 1: done  
  
+-----+  
|⚠ You must enter at least one equipment before finishing. |  
+-----+  
  
📝 Enter the name of selected equipment 1: stand mixer  
  
📝 Enter the name of selected equipment 2: done  
  
Stop adding. Continue to Recipe Instruction page.....
```

Figure 4.6.5.10 Sample output of selecting equipment function

The figure shows that the function will display the equipment stored in the baker equipment file for the user to select. The function can also successfully identify whether the user has entered at least one piece of equipment and carry out the corresponding process.

```

2 usages ± hxlum+1
521 def recipe_instruction():
522     instructions = [] # initialize an empty list to store entered instructions
523
524     category, recipe_name = create_recipe() # get category and recipe name
525     ingredients, ingredient_notes = recipe_ingredient(addng_ingredient=True, update_ingredient=True) # get selected ingredient l
526     equipments = recipe_equipment(update_equipment=True) # get selected equipment list
527
528     # print the selected equipments and ingredients
529     print('')
530     print('-' * 140)
531     print(
532         "\nWelcome to the Recipe Instruction page! Let's get started with creating your delicious bakery goods step by step.\n")
533     print("👉 Selected ingredients 👈")
534     max_length = 0
535     index = 1
536     for item in ingredients:
537         if len(item) > max_length:
538             max_length = len(item)
539
540         print(f'{index}. {item[0].ljust(max_length).title()} x {item[1]} {item[2]}\b')
541         index += 1
542
543     print("\n🌟 Selected equipments 🌟")
544     index = 1
545     for item in equipments:
546         print(f'{index}. {item.title()}\b')
547         index += 1
548
549     if index == len(equipment):
550

```

Figure 4.6.5.11 Function that prompt user to enter instructions, baking information and cost

This is the last part of the add recipe feature, where this function calls and gets return value from `create_recipe()`, `recipe_ingredients()` and `recipe_equipments()` function. It also initializes `adding_ingredient`, `update_ingredient` and `add_equipment` parameters to True. Next, it displays the previously selected ingredients and equipment and asks the user for baking information, starting with the baking temperature, which must be between 0 and 300 degrees Celsius. It then requests the baking time, ensuring it falls between 0 and 90 minutes, and checks that the cost per unit is greater than 0. The user is also required to provide at least one instruction, which is validated before saving the recipe details to the baker recipe file. The function will then ask user if want to continue to add another recipe, if yes, it will recursively called itself, if no, it will call `recipe_management()` function and exit the loop.

Sample Output:

```
Welcome to the Recipe Instruction page! Let's get started with creating your delicious bakery goods step by step.

👉 Selected ingredients 🍉
1. Whole Wheat Flour x 400.0 g
2. Maple Syrup x 20.0 ml

🛠 Selected equipments 🛠
1. Stand Mixer

Please provide the baking temperature (°C): 100

Please provide the baking time (in minutes): 30

Please provide the cost to make one unit of this product (RM): 9

📋 Instructions (type "done" to finish)
1. Mix whole wheat flour, oil, and maple syrup; press into a pan.
2. Blend cream cheese, maple syrup, and eggs; pour over crust.
3. Bake at 350°F for 45 minutes; chill before serving.
4. done

Stop adding instructions... Recipe Cheese Cake successfully saved!

Continue adding new recipe? (y=yes, n=no)
>>> n

Stop adding. Exiting to Recipe Management page.....
```

Figure 4.6.5.12 Sample output of recipe_instruction() function

```

261 },
262 "cheese cake": {
263     "recipe_category": "Cakes",
264     "recipe_name": "cheese cake",
265     "ingredient_used": [
266         [
267             "whole wheat flour",
268             400.0,
269             "g"
270         ],
271         [
272             "maple syrup",
273             20.0,
274             "ml"
275         ]
276     ],
277     "ingredient_notes": "maple syrup must store in fridge",
278     "equipment_used": [
279         "stand mixer"
280     ],
281     "baking_temperature": 100,
282     "baking_time": 30,
283     "cost_per_unit": 9.0,
284     "instructions": [
285         "Mix whole wheat flour, oil, and maple syrup; press into a pan.",
286         "Blend cream cheese, maple syrup, and eggs; pour over crust.",
287         "Bake at 350\u00b0F for 45 minutes; chill before serving."
288     ]
289 }
290 }
```

Figure 4.6.5.13 Sample output of recipe details saved to baker recipe file

Figure 4.6.5.12 shows that the function display correctly the previously selected ingredients and equipment. Next, it prompts user to enter baking information, cost per unit and instructions and validate the input before saving the data. Figure 4.6.5.13 shows the recipe data is successfully saved to baker recipe txt file.

(b) Update recipe

```
659 # define the function to display the details of selected recipe
660 #usage  hxium+1
661 def display_recipe(recipe):
662     # print the details of selected recipe with formatting
663     print(f'{"recipe_category":<20}: {recipe["recipe_category"]}')
664     print(f'{"recipe_name":<20}: {recipe["recipe_name"]}')
665     print(f'\n{"Ingredient Used":>1}')
666     ingredient_index = 1
667     max_length = 0
668     for item in recipe['ingredient_used']:
669         if len(item[0]) > max_length:
670             max_length = len(item[0])
671
672         max_length_unit = 0
673         for items in recipe['ingredient_used']:
674             if len(str(items[1])) > max_length_unit:
675                 max_length_unit = len(str(items[1]))
676
677         print(
678             f'{ingredient_index}. {item[0].ljust(max_length + 2).title()}x {str(item[1]).ljust(max_length_unit + 1)} {item[2]}')
679         ingredient_index += 1
680     print(f'\n{"Ingredient Notes":<20}: {recipe["ingredient_notes"]}')
681     print(f'\n{"Equipment Used":>1}')
682     equipment_index = 1
683     for item in recipe['equipment_used']:
684         print(f'{equipment_index}. {item.title()')
685         equipment_index += 1
686     print(f'\n{"Baking Temperature (°C)":<26}: {recipe["baking_temperature"]}')
687     print(f'\n{"Baking Time (min)":<25}: {recipe["baking_time"]}')
688     print(f'\n{"Cost per Unit (RM)":<25}: {recipe["cost_per_unit"]}')
689     print(f'\n{"Instructions":>1}'
```

Figure 4.6.5.14 Function to display details of selected recipe

This function serves to format the recipe data retrieved from baker recipe file.

Sample Output:

```
-----  
CHEESE CAKE'S DATA  
-----  
recipe_category      : Cakes  
recipe_name          : cheese cake  
  
🍏 ingredient_used:  
1. Whole Wheat Flour  x  400.0  g  
2. Maple Syrup        x  20.0   ml  
  
ingredient_notes     : maple syrup must store in fridge  
  
🛠 equipment_used:  
1. Stand Mixer  
  
baking_temperature (°C)  : 100  
baking_time (min)        : 30  
cost_per_unit (RM)       : 9.0  
  
instructions:  
1. Mix whole wheat flour, oil, and maple syrup; press into a pan.  
2. Blend cream cheese, maple syrup, and eggs; pour over crust.  
3. Bake at 100 degree celsius for 30 minutes; chill before serving.
```

Figure 4.6.5.15 Sample output of formatted details of selected recipe

The output shows the formatted details of selected recipe.

```

696     def update_recipe():
697         while True:
698             index = 1
699             printed_centered('RECIPE LIST')
700             for key, recipe in recipe_data.items():
701                 print(f'{index}. {recipe["recipe_name"].title()}') # print the index with recipe name
702                 index += 1
703             print(f'{len(recipe_data) + 1}. Cancel')
704
705             # prompt user to select a recipe to edit
706             try:
707                 index_of_product_to_edit = int(
708                     input(
709                         f'\nEnter index number to update the information of the recipe (or enter {len(recipe_data) + 1} to cancel):\n>>> ')
710
711                 if index_of_product_to_edit == len(recipe_data) + 1: # if "Cancel" is selected
712                     print('\nCancelling. Exiting to Product Management page.....') # return to the previous page
713                     recipe_management()
714                     break
715
716                 # id selected index falls within valid range
717                 elif 1 <= index_of_product_to_edit <= len(recipe_data):
718                     for key, recipe in recipe_data.items():
719                         selected_recipe_key = list(recipe_data.keys())[index_of_product_to_edit - 1] # indicate which recipe is selected
720                         break
721
722                 else:
723                     print('\n! Recipe not found.') # error displayed if selected product not found
724                     continue
725
726             except ValueError:
727                 print('Please enter a valid integer value.')

```

Figure 4.6.5.16 Function of update selected recipe

The function begins by displaying a list of recipes stored in baker recipe file with their corresponding index numbers. Users can select a recipe to edit by entering its index number or enter the last index number to cancel the process. If a valid index is provided, the function retrieves the selected recipe data and calls display_recipe() function to format and display the recipe details. Next, it will prompt user to enter the selected attribute they wish to modify. The user input is validated to ensure it exists in selected recipe data.

For ingredient modification, users can add, edit or delete items within the ingredient used list. If they choose to add an ingredient, the function calls recipe_ingredient() function and initializes the adding_ingredient parameter to True, and update_ingredient to False, to capture new ingredient details, which are then appended to the ingredient list and saved. For editing, user needs to enter the index number of selected ingredients, and the function will call recipe_ingredient() function and initializes both adding_ingredient and update_ingredient parameter to False. The function will replace the existing ingredient with new details returned from recipe_ingredient() function, while deletion removes the selected ingredient based on its index.

Similar functionality applies to equipment_used attribute, user can add new equipment, edit existing equipment and delete items. If they choose to add an ingredient, the function calls recipe_equipment() function and initializes the add_equipment parameter to True to capture

new equipment details until user input ‘done’ to stop adding, which are then appended to the equipment list and saved. For editing, user needs to enter the index number of selected equipment, and the function will call `recipe_equipment()` function and initializes `add_equipment` parameter to False. The function will replace the existing equipment with new details returned from `recipe_equipment()` function, while deletion removes the selected equipment based on its index.

To modify the instructions attribute, the function will display the current instructions and prompt user to enter new instruction. Users can enter each step sequentially and type ‘done’ when want to stop the process. At least one instruction is required to complete the update. If user have entered at least one instruction and type ‘done’, the function will replace the original instruction with the new value and save the data.

To modify general attributes (other than ingredient used, equipment used and instructions), the function will prompt user to enter a new value and validate it based on requirement of selected attribute. If the input is valid, it will update the old data with new value and save it.

Sample Output:

```
-----  
          RECIPE LIST  
-----  
1. Tiramisu  
2. Bello  
3. Matcha Layer Cake  
4. Vanilla Muffin  
5. Applepie  
6. Red Velvet  
7. Garlic Bun  
8. Swiss Roll  
9. Cheese Cake  
10. Cancel  
  
Enter index number to update the information of the recipe (or enter 10 to cancel):  
>>> 9
```

Figure 4.6.5.17 Sample output of recipe list

The figure shows that the function retrieve recipe data from baker recipe file and display them with corresponding index number. The function also validates the user input, ensure it is valid before proceeding to next step.

```
-----  
CHEESE CAKE'S DATA  
-----  
  
recipe_category      : Cakes  
recipe_name          : cheese cake  
  
👉 ingredient_used:  
1. Whole Wheat Flour x 400.0 g  
2. Maple Syrup       x 20.0 ml  
  
ingredient_notes     : maple syrup must store in fridge  
  
✖ equipment_used:  
1. Stand Mixer  
  
baking_temperature (°C) : 100  
baking_time (min)       : 30  
cost_per_unit (RM)      : 9.0  
  
instructions:  
1. Mix whole wheat flour, oil, and maple syrup; press into a pan.  
2. Blend cream cheese, maple syrup, and eggs; pour over crust.  
3. Bake at 100 degree celsius for 30 minutes; chill before serving.  
  
To change the information, please enter the exact matching name but without the (). Example: baking_time.  
Which information do you want to update? (or enter "cancel")  
=> Ingredient_used
```

Figure 4.6.5.18 Sample output of selected recipe details

The figure shows that after ensuring the input index is valid, the function will display the recipe details of selected recipe correctly. Next, it will prompt user to input an attribute to update. If user input does not find in the given recipe data, it will display error message and prompt user to enter again.

```

1. Whole Wheat Flour  x  400.0  g
2. Maple Syrup        x  20.0  ml

Add, edit or delete ingredient?
Please enter 'add', 'edit' or 'delete' (or enter 'cancel' to stop): add

-----
          INGREDIENT LIST
-----

  ♦ Flours and Grains ♦
1. Whole Wheat Flour
2. Cornmeal
3. Almond Flour

  ♦ Sweeteners ♦|
1. Maple Syrup
2. Candy

  ♦ Dairy and Non-Dairy Products ♦
1. Whipping Cream

  ♦ Fruits and Vegetables ♦
1. Apple

  ♦ Spices and Flavourings ♦
1. Chilly

  ♦ Leavening Agents ♦
1. Baking Powder

```

Figure 4.6.5.19 Sample output when selected attribute is ingredient used (1)

```

👉 Enter the ingredient name: almond flour

👉 Allowable unit measurement: g, kg
👉 Enter the unit measurement of almond flour: g

👉 Enter the quantity per unit of almond flour: 60

Ingredient so far:
1. Almond Flour  x  60.0  g

Continue adding ingredients? (y=yes, n=no)
>>> n

Information saved.

```

Figure 4.6.5.20 Sample output when selected attribute is ingredient used (2)

Figure 4.6.5.19 and Figure 4.6.5.20 shows that the function will display the existing ingredient and prompt user to choose whether want to add, edit or delete the ingredient used. The output shows the scenario when user want to add ingredients. The function will display the available ingredient list and prompt user to select an ingredient, enter its unit measurement and quantity. After validating the input, it will add the new ingredient details to the ingredient list and save data.

```
To change the information, please enter the exact matching name but without the (). Example: baking_time.
Which information do you want to update? (or enter "cancel")
>>> equipment_used

1. Stand Mixer
2. Convection Oven

Add, edit or delete equipment?
Please enter 'add', 'edit' or 'delete' (or enter 'cancel' to stop): edit

Please enter the index number of equipment you want to edit (or enter 'cancel')
>>> 2
-----
          EQUIPMENT LIST
-----
  ↗ Ovens and Baking Appliances ↗
1. Convection Oven
2. Toaster Oven

  ↗ Mixing Equipment ↗
1. Stand Mixer

💡 Please enter the name of selected equipment (or type "done" to finish)

✍ Enter the name of selected equipment 1: toaster oven

Information saved.
```

Figure 4.6.5.21 Sample output when selected attribute is equipment used

Figure 4.6.5.21 shows that the function will display the existing equipment with corresponding index and prompt user to choose whether want to add, edit or delete the equipment used. The output shows the scenario when user want to edit equipment. Users need to enter the index number of selected equipment. Once the input is validated, the function will display the available equipment list and prompt user to select an equipment. After validating the input, it will replace the old data with new equipment value and save data.

```
To change the information, please enter the exact matching name but without the (). Example: baking_time.
Which information do you want to update? (or enter "cancel")
>>> instructions
1. Mix Whole Wheat Flour, Oil, And Maple Syrup; Press Into A Pan.
2. Blend Cream Cheese, Maple Syrup, And Eggs; Pour Over Crust.
3. Bake At 100 Degree Celsius For 30 Minutes; Chill Before Serving.

Enter new instruction. (type "done" to finish updating, "cancel" to exit to previous page.)
1. Bake crust: flour, butter, sugar.
2. Mix cream cheese, maple syrup, sugar, eggs.
3. Pour, bake 50 mins. Chill.
4. Drizzle maple syrup.
5. done

Information saved.
```

Figure 4.6.5.22 Sample output when selected attribute is instruction

Figure 4.6.5.22 shows that the function will display the existing instructions and prompt user to enter new instruction. User can type ‘done’ if wish to stop the process. Once the input is validated to ensure have at least one value, the function will replace the old instructions with new value and save data.

```
To change the information, please enter the exact matching name but without the (). Example: baking_time.
Which information do you want to update? (or enter "cancel")
>>> baking_time

Enter new baking_time: 50

baking_time of cheese cake is updated.
```

Figure 4.6.5.23 Sample output when general attribute is selected

Figure 4.6.5.23 shows the scenario when general attribute is selected. The function will prompt user to enter a new value and validate it based on requirement of selected attribute. If the input is valid, it will update the old data with new value and save it.

```
-----  
CHEESE CAKE'S DATA  
-----  
recipe_category      : Cakes  
recipe_name          : cheese cake  
  
apple ingredient_used:  
1. Whole Wheat Flour x 400.0 g  
2. Maple Syrup       x 20.0 ml  
3. Almond Flour      x 60.0 g  
  
ingredient_notes     : maple syrup must store in fridge  
  
star equipment_used:  
1. Stand Mixer  
2. Toaster Oven  
  
baking_temperature (°C) : 100  
baking_time (min)       : 50  
cost_per_unit (RM)      : 9.0  
  
instructions:  
1. Bake crust: flour, butter, sugar.  
2. Mix cream cheese, maple syrup, sugar, eggs.  
3. Pour, bake 50 mins. chill.  
4. Drizzle maple syrup.
```

Figure 4.6.5.24 Sample output of updated recipe details

```

265     "cheese cake": {
266         "recipe_category": "Cakes",
267         "recipe_name": "cheese cake",
268         "ingredient_used": [
269             [
270                 "whole wheat flour",
271                 400.0,
272                 "g"
273             ],
274             [
275                 "maple syrup",
276                 20.0,
277                 "ml"
278             ],
279             [
280                 "almond flour",
281                 60.0,
282                 "g"
283             ]
284         ],
285         "ingredient_notes": "maple syrup must store in fridge",
286         "equipment_used": [
287             "stand mixer",
288             "toaster oven"
289         ],
290         "baking_temperature": 100,
291         "baking_time": 50,
292         "cost_per_unit": 9.0,
293         "instructions": [
294             "Bake crust: flour, butter, sugar.",
295             "Mix cream cheese, maple syrup, sugar, eggs.",
296             "Pour, bake 50 mins. Chill."

```

Figure 4.6.5.25 Sample output of updated recipe details in text file

Figure 4.6.4.24 and Figure 4.6.4.25 shows that the recipe data is successfully updated and saved in baker recipe file.

(c) Delete recipe

```
1143 def delete_recipe():
1144     while True:
1145         index = 1
1146         print('\n-----')
1147         print('\t\t\t\t', 'RECIPE LIST')
1148         print('-----')
1149         for name, info in recipe_data.items():
1150             print(f'{index}. {name.title()}') # print recipe name with its index
1151             index += 1
1152         print(f'{len(recipe_data) + 1}. cancel')
1153
1154     try:
1155         index_remove_recipe = int(
1156             input(f'\nWhich recipe do you want to remove? (or enter {len(recipe_data) + 1} to cancel)\n>> '))
1157         if index_remove_recipe == len(recipe_data) + 1: # if user choose 'cancel'
1158             print('\nCancelling. Exiting to Services page.....')
1159             recipe_management() # return to recipe management page
1160             break
1161
1162         # if selected index falls within valid range
1163         elif 1 <= index_remove_recipe <= len(recipe_data):
1164             recipe_to_remove = list(recipe_data.keys())[index_remove_recipe - 1] # identify recipe to remove by accessing the index of key of recipe
1165             del recipe_data[recipe_to_remove] # delete the selected recipe
1166             save_info(recipe_data) # save the data
1167             print(
1168                 f'\n{recipe_to_remove.title()} is removed.' # inform user that the selected recipe is removed successfully
1169
1170             # prompt user whether to remove another recipe
1171
```

Figure 4.6.5.26 Function for delete recipe

The function begins by displaying the recipe stored in baker recipe file and assign them a numbered index for user to choose. The user input index is validated so that it falls within the valid range. User can also choose the last index number to cancel the update process. If user choose the cancel option, they will return to recipe management page. If the selected index is valid, the function identifies the corresponding product by converting the index to the recipe data key. Next, it removes the product data based on the recipe data key and save the data.

After deletion, the function prompt user whether to continue delete another recipe. If user choose yes, the deletion process continues from the beginning of function. If user choose no, it will exit the loop and call `recipe_management()` function to return to recipe management menu. If the input is invalid it will prompt user to enter again.

Sample Output:

```
-----  
          RECIPE LIST  
-----  
1. Tiramisu  
2. Bello  
3. Matcha Layer Cake  
4. Vanilla Muffin  
5. Applepie  
6. Red Velvet  
7. Garlic Bun  
8. Swiss Roll  
9. Cheese Cake  
10. cancel  
  
Which recipe do you want to remove? (or enter 10 to cancel)  
>>> 9  
  
Cheese Cake is removed.  
  
Continue to remove? (y=yes, n=no)  
>>> n  
  
Stop removing. Exiting to Recipe Management page.....  
-----  
          RECIPE MANAGEMENT  
-----
```

Figure 4.6.5.28 Sample output of recipe delete process

The figure shows that the function retrieve recipe data from baker recipe file and display them with corresponding index number. After ensuring the input is valid, it will remove the data of selected recipe and display a confirmation message. Next, it will prompt user whether to continue delete another recipe and carry out corresponding process based on user's input.

5.0 Conclusion

In conclusion, the bakery management system is a reliable and practical solution for daily operational needs. It can solve the key challenges that they are facing by automating the system administration, order management, financial tracking and so on. The automation allows the bakery's operation to run smoothly and efficiently (Aurelio, 2024) at the same time attract and maintain customer loyalty. With the system, manager can manage the inventory and oversee profitability report conveniently. Bakers can report the malfunction equipment to manager in time to ensure daily productions. Cashier can easily generate reports on the popularity sales that help in satisfying the preference of customers. Customers can purchase products by viewing the digital menu and get digital receipts after payment. Overall, the system provides valuable insights to allow business growth and ensure its sustainability (Anon, 2024).

6.0 References

Anon. (2019). What are the pros and cons of manual inventory systems? *Quora*.

<https://www.quora.com/What-are-the-pros-and-cons-of-manual-inventory-systems>

Anon. (2024). The rise of bakery management software: Transforming operations in the digital

age. *Market Research Intellect*. <https://www.marketresearchintellect.com/blog/the-rise-of-bakery-management-software-transforming-operations-in-the-digital-age/>

Aurelio, S. (2024). Top benefits of using inventory management for bakeries. *Itemit*.

<https://itemit.com/top-benefits-of-using-inventory-management-software-for-bakeries/>

MuftaSoft. (2023). Pros and cons of bakery automation. <https://www.linkedin.com/pulse/pros-cons-bakery-automation-muftasoft-ymxqf/>

7.0 Workload matrix

TP number	Name	Role	Documentation
TP076390	Ng Yvonne	<ul style="list-style-type: none"> • Manager • Part of cashier (transaction completion) 	<ul style="list-style-type: none"> • All part of manager • Transaction completion of cashier • Conclusion
TP076160	Lum Han Xun	<ul style="list-style-type: none"> • Baker • Part of cashier (product display & reporting) 	<ul style="list-style-type: none"> • All part of baker • Sign in/Login, Product display and reporting of cashier
TP077232	Heng Xin Hui	<ul style="list-style-type: none"> • Customer • Part of cashier (discount management) 	<ul style="list-style-type: none"> • Introduction • All part of customer • Discount management of cashier