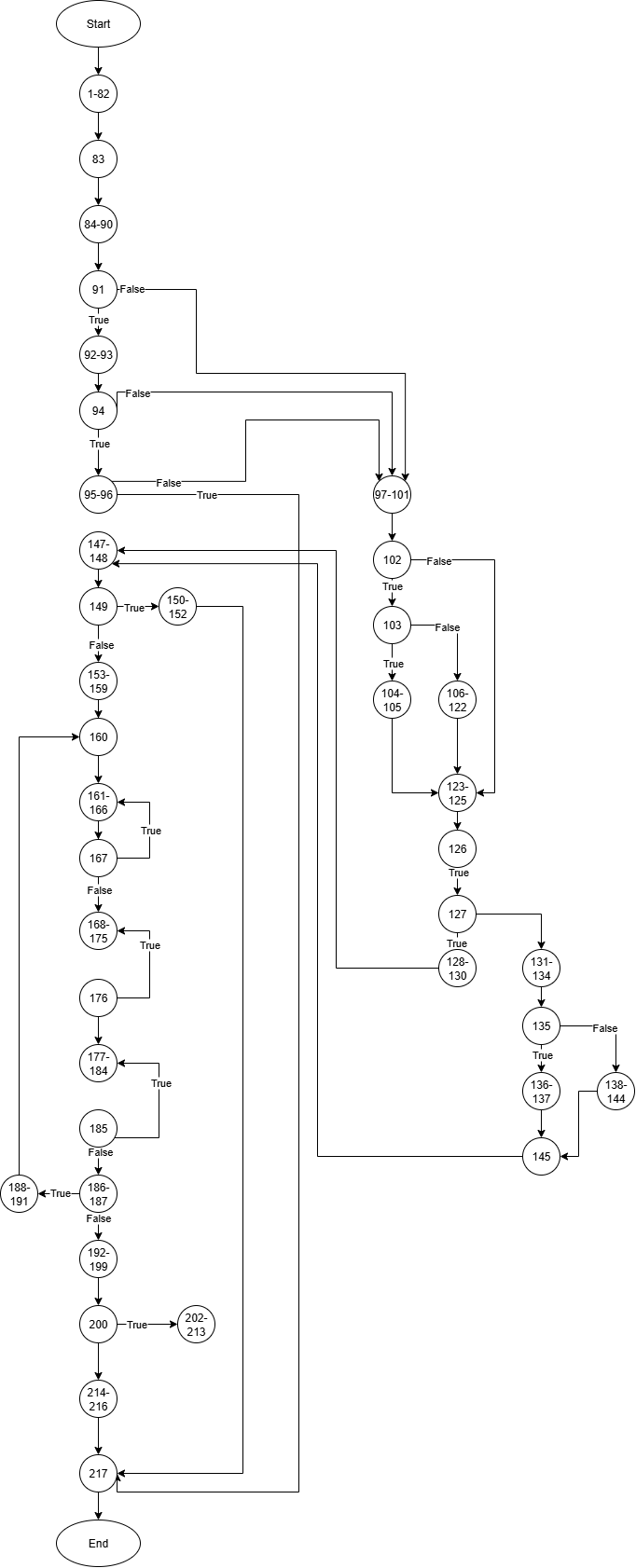
**LAB 4: Re-write and extend the program Enrol\_Gen.c with name Gen\_Enrol\_Safe\_Input for the following additional functionalities:**

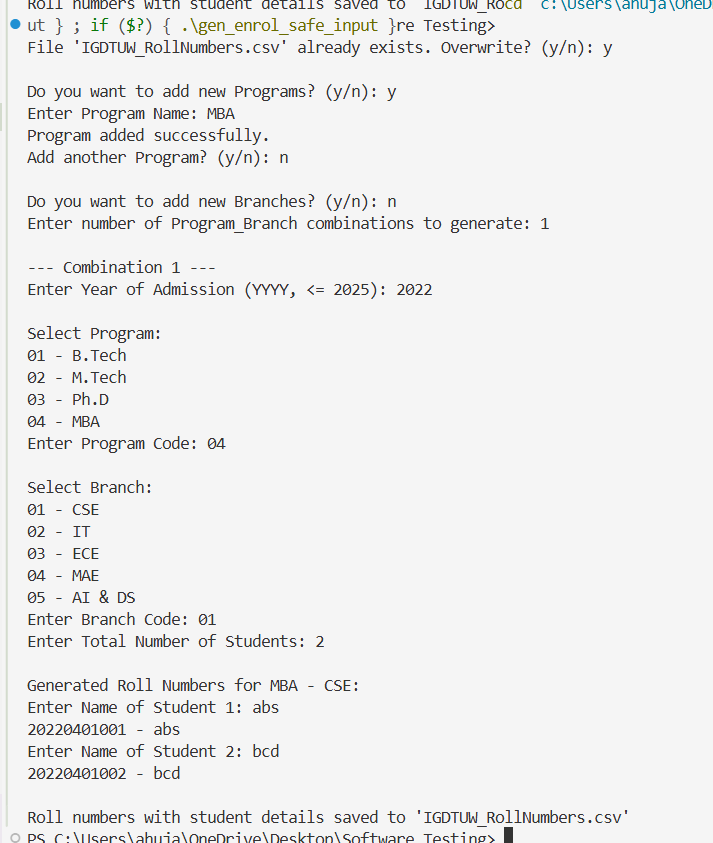
1. User can interactively add Program and/or Course for enrollment generation without duplicity of Branch and/or Course.
2. The output shall be enrollment numbers, Name of the student, Program and course for each course in the program

4.1: **Prepare the CFG for the program Gen\_Enrol\_Safe\_Input.c**



4.2: **Write test cases to find the error in the program**

1. **Test Case1**:

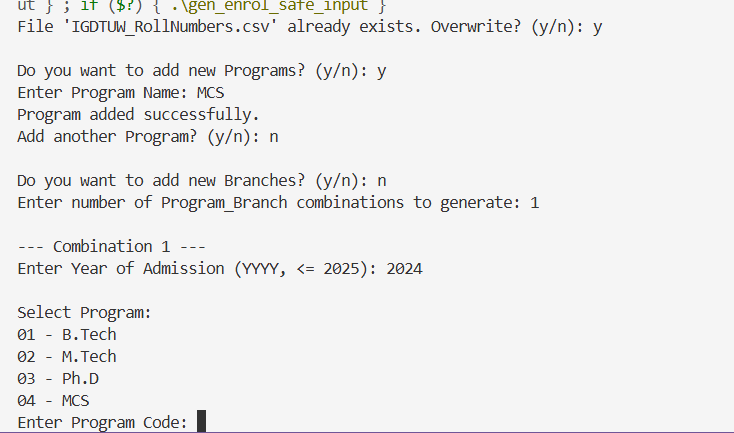


Expected Output: Update the programs List.

Actual Output: Updated the programs list.

Result: PASS

2. **Test Case2:**

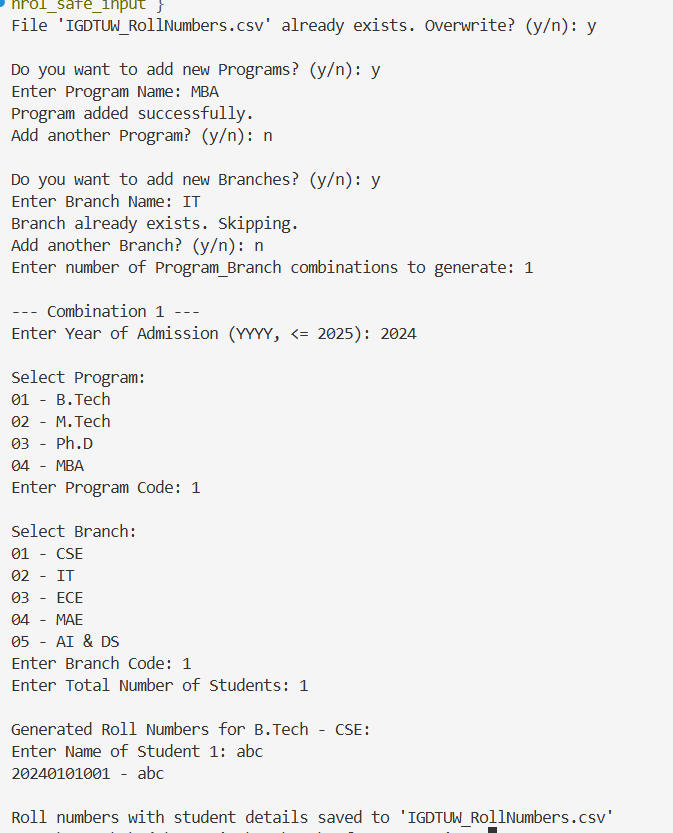


Expected Output: New identifier allotted to newly added program.

Actual Output: It overwrote the previous updates.

Result: FAIL

3. **Test Case 3**:

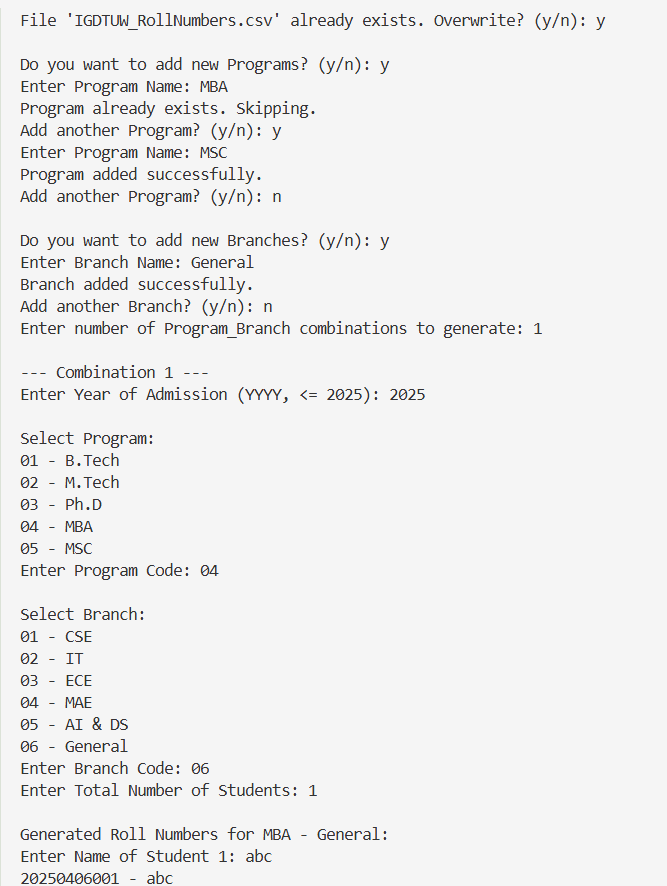


Expected Output: It should have skipped the previously added program.

Actual Output: It still added the program as new one.

Result: FAIL

4. **Test Case4:**

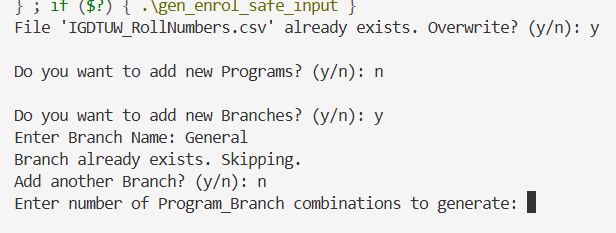


Expected Output: Updating the branch list.

Actual Output: Correctly updated the branch list.

Result: PASS

5. **Test Case 5:**



Expected Output: Skip adding the already existing branch.

Actual Output: Skipped the already existing branch.

Result: PASS

4.3: **Summarize the errors to facilitate the user to debug the program**

The code is not able to handle these cases:

* Maintaining the updated programs list, whenever we re run the program it loses all the updates.
* Maintaining updated branch list.
* overwriting the id associated with previously added program or branch list.

4.4: **Debug the program for the identified error**

#include <stdio.h>

#include <string.h>

#include <time.h>

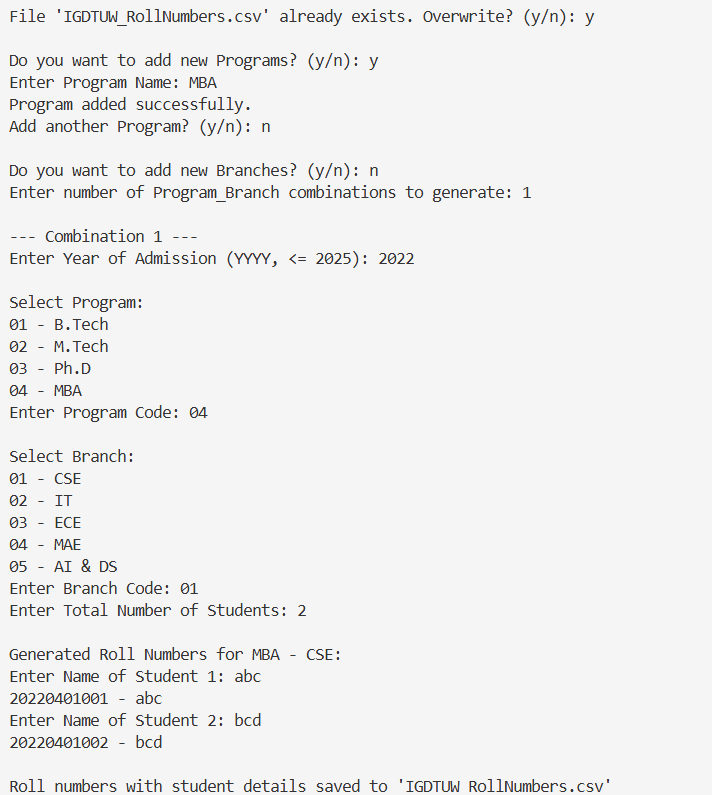
#define **MAX\_PROGRAMS** 20

#define **MAX\_BRANCHES** 20

1. typedef struct {
2. int code;
3. char name[50];
4. } **Program**;
5. typedef struct {
6. int code;
7. char name[50];
8. } **Branch**;
9. *// ---------- File Handling ----------*
10. int **loadPrograms**(**Program** programs[], const char \*filename) {
11. **FILE** \*fp  = **fopen**(filename, "r");
12. int count = 0;
13. if(fp){
14. while(**fscanf**(fp, "%d %[^\n]", &programs[count].code, programs[count].name) == 2){
15. count++;
16. }
17. **fclose**(fp);
18. }
19. return count;
20. }
21. void **saveProgram**(const **Program** \*p, const char \*filename){
22. **FILE** \*fp = **fopen**(filename, "a");
23. if(fp){
24. **fprintf**(fp, "%d %s\n", p->code, p->name);
25. **fclose**(fp);
26. }
27. }
28. int **loadBranches**(**Branch** branches[], const char \*filename) {
29. **FILE** \*fp = **fopen**(filename, "r");
30. int count = 0;
31. if (fp) {
32. while (**fscanf**(fp, "%d %[^\n]", &branches[count].code, branches[count].name) == 2) {
33. count++;
34. }
35. **fclose**(fp);
36. }
37. return count;
38. }
39. void **saveBranch**(const **Branch** \*b, const char \*filename) {
40. **FILE** \*fp = **fopen**(filename, "a");
41. if (fp) {
42. **fprintf**(fp, "%d %s\n", b->code, b->name);
43. **fclose**(fp);
44. }
45. }
46. int **fileExists**(const char \*filename) {
47. **FILE** \*file = **fopen**(filename, "r");
48. if (file) {
49. **fclose**(file);
50. return 1;
51. }
52. return 0;
53. }
54. int **getCurrentYear**() {
55. **time\_t** t = **time**(**NULL**);
56. struct **tm** tm = \***localtime**(&t);
57. return tm.tm\_year + 1900;
58. }
59. int **combinationExists**(int used[][2], int usedCount, int progCode, int branchCode) {
60. for (int i = 0; i < usedCount; i++) {
61. if (used[i][0] == progCode && used[i][1] == branchCode) {
62. return 1;
63. }
64. }
65. return 0;
66. }
67. int **programExists**(**Program** programs[], int numPrograms, const char \*name) {
68. for (int i = 0; i < numPrograms; i++) {
69. if (**strcmp**(programs[i].name, name) == 0) {
70. return 1;
71. }
72. }
73. return 0;
74. }
75. int **branchExists**(**Branch** branches[], int numBranches, const char \*name) {
76. for (int i = 0; i < numBranches; i++) {
77. if (**strcmp**(branches[i].name, name) == 0) {
78. return 1;
79. }
80. }
81. return 0;
82. }
83. int **main**() {
84. **Program** programs[**MAX\_PROGRAMS**];
85. int numPrograms = **loadPrograms**(programs, "programs.txt");
86. **Branch** branches[**MAX\_BRANCHES**];
87. int numBranches = **loadBranches**(branches, "branches.txt");
88. char filename[] = "IGDTUW\_RollNumbers.csv";
89. char choice;
90. *// Check if file exists*
91. if (**fileExists**(filename)) {
92. **printf**("File '%s' already exists. Overwrite? (y/n): ", filename);
93. **scanf**(" %c", &choice);
94. if (choice != 'y' && choice != 'Y') {
95. **printf**("Operation cancelled. File not overwritten.\n");
96. return 0;
97. }
98. }
99. *//Add new Programs*
100. **printf**("\nDo you want to add new Programs? (y/n): ");
101. **scanf**(" %c", &choice);
102. while (choice == 'y' || choice == 'Y') {
103. if (numPrograms >= **MAX\_PROGRAMS**) {
104. **printf**("Maximum program limit reached.\n");
105. break;
106. }
107. char progName[50];
108. **printf**("Enter Program Name: ");
109. **scanf**(" %[^\n]", progName);
110. if (**programExists**(programs, numPrograms, progName)) {
111. **printf**("Program already exists. Skipping.\n");
112. } else {
113. **Program** p;
114. p.code = (numPrograms == 0) ? 1 : programs[numPrograms - 1].code + 1;
115. **strcpy**(p.name, progName);
116. programs[numPrograms++] = p;
117. **saveProgram**(&p, "programs.txt");
118. **printf**("Program added successfully.\n");
119. }
120. **printf**("Add another Program? (y/n): ");
121. **scanf**(" %c", &choice);
122. }
123. *//Add new Branches*
124. **printf**("\nDo you want to add new Branches? (y/n): ");
125. **scanf**(" %c", &choice);
126. while (choice == 'y' || choice == 'Y') {
127. if (numBranches >= **MAX\_BRANCHES**) {
128. **printf**("Maximum branch limit reached.\n");
129. break;
130. }
131. char branchName[50];
132. **printf**("Enter Branch Name: ");
133. **scanf**(" %[^\n]", branchName);
134. if (**branchExists**(branches, numBranches, branchName)) {
135. **printf**("Branch already exists. Skipping.\n");
136. } else {
137. **Branch** b;
138. b.code = (numBranches == 0) ? 1 : branches[numBranches - 1].code + 1;
139. **strcpy**(b.name, branchName);
140. branches[numBranches++] = b;
141. **saveBranch**(&b, "branches.txt");
142. **printf**("Branch added successfully.\n");
143. }
144. **printf**("Add another Branch? (y/n): ");
145. **scanf**(" %c", &choice);
146. }
147. **FILE** \*fp = **fopen**(filename, "w");
148. if (!fp) {
149. **printf**("Error: Could not create file.\n");
150. return 1;
151. }
152. **fprintf**(fp, "Year,ProgramCode,ProgramName,BranchCode,BranchName,RollNumber,StudentName\n");
153. int combinations;
154. **printf**("Enter number of Program\_Branch combinations to generate: ");
155. **scanf**("%d", &combinations);
156. int usedCombinations[100][2];
157. int usedCount = 0;
158. int currentYear = **getCurrentYear**();
159. for (int c = 1; c <= combinations; c++) {
160. int year, progChoice, branchChoice, totalStudents;
161. **printf**("\n--- Combination %d ---\n", c);
162. *// Year*
163. do {
164. **printf**("Enter Year of Admission (YYYY, <= %d): ", currentYear);
165. **scanf**("%d", &year);
166. } while (year > currentYear);
167. *// Program*
168. do {
169. **printf**("\nSelect Program:\n");
170. for (int j = 0; j < numPrograms; j++) {
171. **printf**("%02d - %s\n", programs[j].code, programs[j].name);
172. }
173. **printf**("Enter Program Code: ");
174. **scanf**("%d", &progChoice);
175. } while (progChoice < 1 || progChoice > numPrograms);
176. *// Branch*
177. do {
178. **printf**("\nSelect Branch:\n");
179. for (int j = 0; j < numBranches; j++) {
180. **printf**("%02d - %s\n", branches[j].code, branches[j].name);
181. }
182. **printf**("Enter Branch Code: ");
183. **scanf**("%d", &branchChoice);
184. } while (branchChoice < 1 || branchChoice > numBranches);
185. *// Duplicate check*
186. if (**combinationExists**(usedCombinations, usedCount, progChoice, branchChoice)) {
187. **printf**("This Program-Branch combination already used. Skipping.\n");
188. c--;
189. continue;
190. }
191. usedCombinations[usedCount][0] = progChoice;
192. usedCombinations[usedCount][1] = branchChoice;
193. usedCount++;
194. *// Students*
195. **printf**("Enter Total Number of Students: ");
196. **scanf**("%d", &totalStudents);
197. **printf**("\nGenerated Roll Numbers for %s - %s:\n",
198. programs[progChoice - 1].name, branches[branchChoice - 1].name);
199. for (int serial = 1; serial <= totalStudents; serial++) {
200. char studentName[50];
201. **printf**("Enter Name of Student %d: ", serial);
202. **scanf**(" %[^\n]", studentName);
203. **printf**("%04d%02d%02d%03d - %s\n",
204. year, progChoice, branchChoice, serial, studentName);
205. **fprintf**(fp, "%d,%02d,%s,%02d,%s,%04d%02d%02d%03d,%s\n",
206. year, progChoice, programs[progChoice - 1].name,
207. branchChoice, branches[branchChoice - 1].name,
208. year, progChoice, branchChoice, serial, studentName);
209. }
210. }
211. **fclose**(fp);
212. **printf**("\nRoll numbers with student details saved to '%s'\n", filename);
213. return 0;
214. }

4.5: **Prepare the set of Test Cases to Re-test the program**.

1. **Test Case1**:

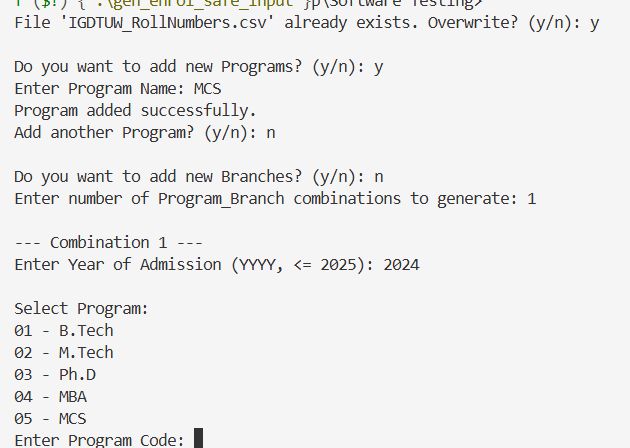


Expected Output: Update the programs list.

Actual Output: Updated the programs list.

Result: PASS

2. **Test Case 2:**

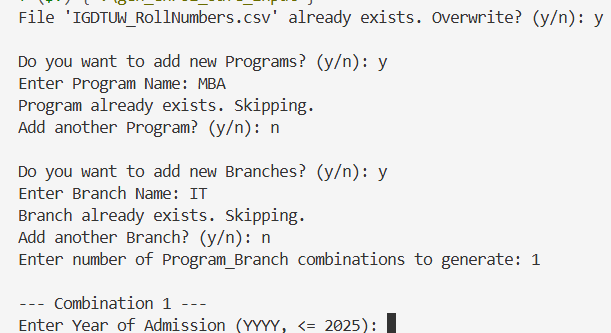


Expected Output: New identifier allotted to new program added.

Actual Output: It appended the new program in the programs list with the new identifier.

Result: PASS

3. **Test Case 3**:

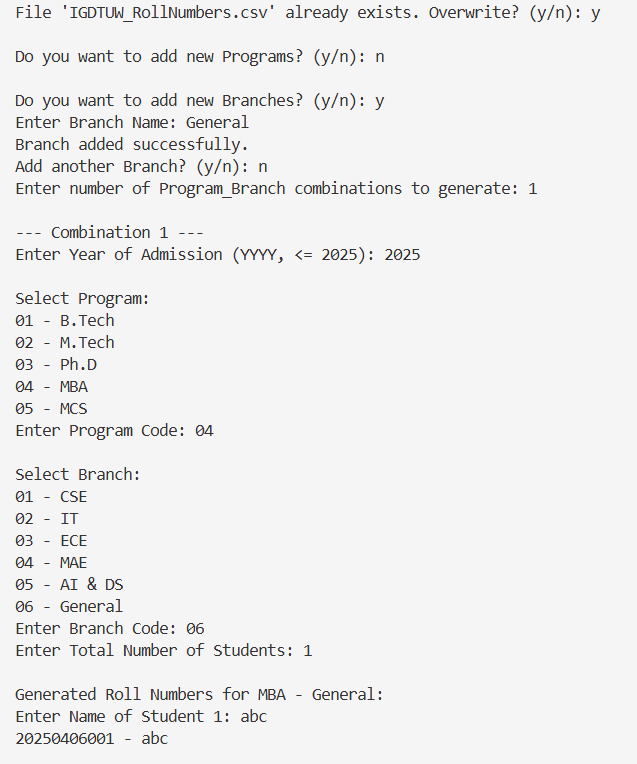


Expected Output: It should have skipped the previously added program.

Actual Output: It skips the already existing programs.

Result: PASS

4. **Test Case 4:**

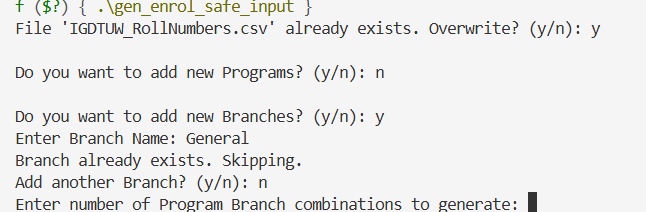


Expected Output: Updating the branch list.

Actual Output: Correctly updated the branch list.

Result: PASS

5**. Test Case 5**:



Expected Output: Skip adding the already existing branch

Actual Output: Skipped the already exisiting branch

Result: PASS