



PHONE DIRECTORY

16010122034 - peeth chowdhary

16010122041 - rohit deshpane

16010122074 - eeshanya joshi

PROBLEM DEFINITION

Creating a phone directory using hashing and linked lists in C. The user will be able to add multiple records of people's names their phone numbers. The user will also be able to search for a record in the given hash table and the extending linked lists.



DATA STRUCTURES USED

- Double Hashing

In our phone directory application, we've implemented a highly efficient double hashing technique for managing records based on last names. This approach is renowned for its speed when it comes to data insertion within an array.



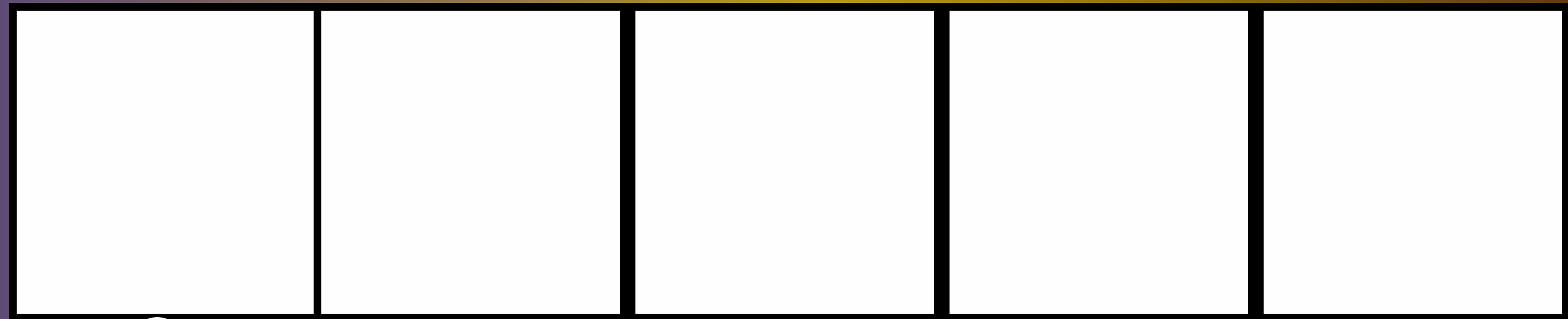
DATA STRUCTURES USED

- **Linked Lists**

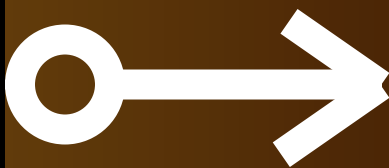
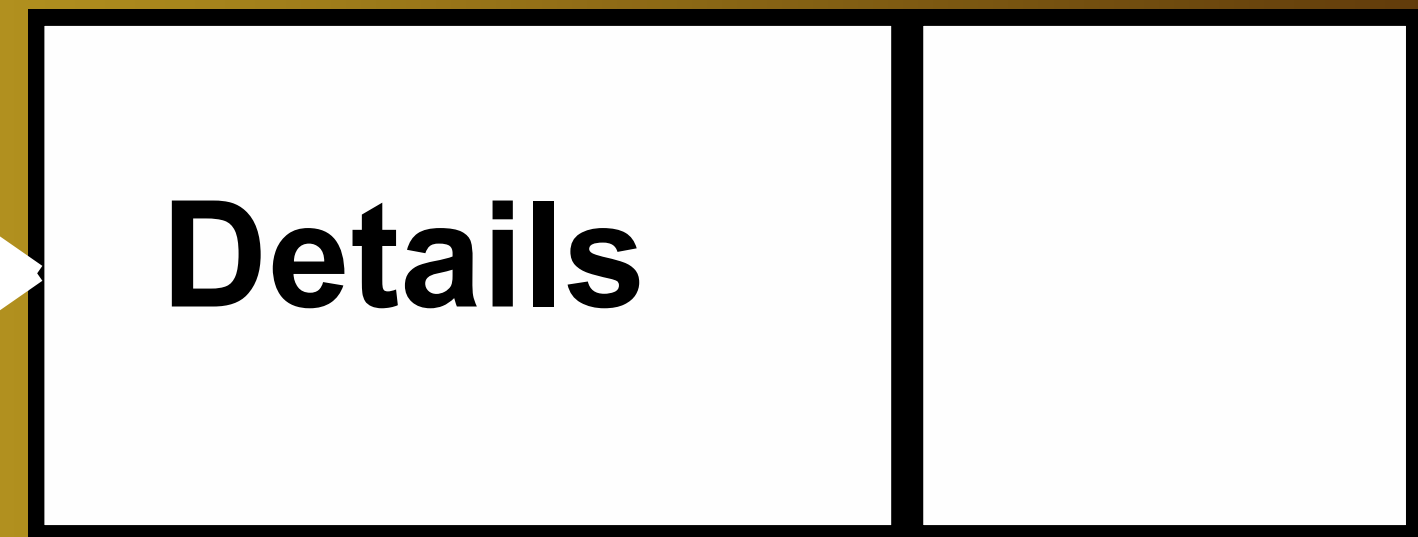
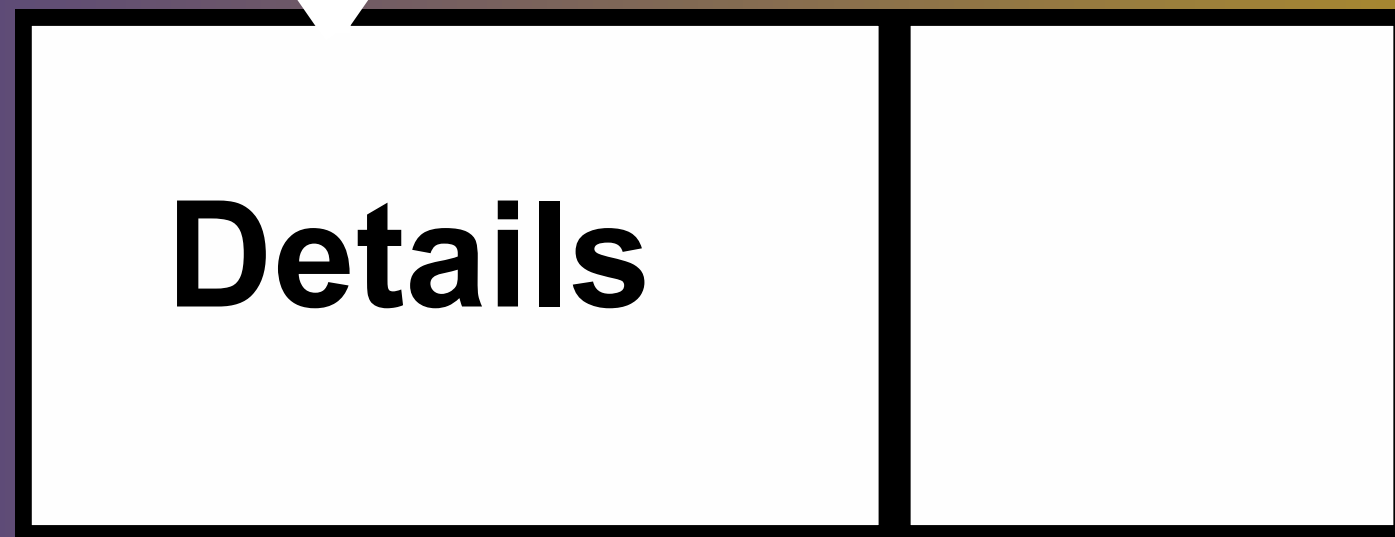
Additionally, in our application, linked lists come into play when all available hashing possibilities are exhausted or a same name is encountered. At this point, the system initiates the creation of nodes to house the record details within a linked list structure.



Hash Table



DATA STRUCTURES USED



Linked List starting from index 0

IMPLEMENTATION DETAILS

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4
5 #define LIST_LEN 3
6 #define STRING_SIZE 50
7
8 struct record* new_record(char* last_name, char* first_name);
9 struct record* append_record(int index, char* last_name, char* first_name);
10 unsigned int find_hash(char* s);
11 unsigned int find_double_hash(char* s, int collisions, unsigned int hash);
12 void add_record(char* last_name, char* first_name);
13 void display_record(char* last_name, char* first_name);
14
15 struct record{
16     int num;
17     char first_name[STRING_SIZE];
18     char last_name[STRING_SIZE];
19     struct record* next;
20 };
21
22 int num_terms = 0;
23 struct record* records[LIST_LEN];
24
25 struct record* new_record(char* last_name, char* first_name){
26     struct record* rec = (struct record*)malloc(sizeof(struct record));
27
28     strncpy(rec->last_name, last_name, STRING_SIZE);
29     strncpy(rec->first_name, first_name, STRING_SIZE);
30
31     printf("Enter the Person's Number: ");
32     scanf("%d", &rec->num);
33
34     return rec;
35 }
```

```

124 unsigned int find_hash(char* s) {
125     unsigned int hash = 0;
126     while (*s) {
127         hash = (hash * LIST_LEN) + *s;
128         s++;
129     }
130     return hash % LIST_LEN;
131 }
133 unsigned int find_double_hash(char* s, int collisions, unsigned int hash) {
134     return (hash + collisions * (1 + (hash % (LIST_LEN - 1))) + collisions) % LIST_LEN;
135 }

```

```

137 void add_record(char* last_name, char* first_name){
138     int index = find_hash(last_name);
139     int collisions = 1;
140
141     if(num_terms == LIST_LEN){
142         int original_index = index;
143         while(strcmp(records[index]->last_name, last_name) != 0){
144             if(collisions == LIST_LEN){
145                 records[index] = append_record(original_index, last_name, first_name);
146                 return;
147             }
148             index = find_double_hash(last_name, collisions, index) % LIST_LEN;
149             ++collisions;
150         }
151         records[index] = append_record(index, last_name, first_name);
152         return;
153     }
154
155     while(records[index] != NULL){
156         if(strcmp(records[index]->last_name, last_name) == 0){
157             records[index] = append_record(index, last_name, first_name);
158             return;
159         }
160         index = find_double_hash(last_name, collisions, index) % LIST_LEN;
161         ++collisions;
162     }
163
164     struct record* rec = new_record(last_name, first_name);
165     records[index] = rec;
166     printf("\nRecord added Successfully.");
167     num_terms++;
168 }

```

```

87 struct record* append_record(int index, char* last_name, char* first_name){
88     struct record* rec = (struct record*)malloc(sizeof(struct record));
89
90     struct record* start = records[index];
91     if(strcmp(start->last_name, last_name) == 0 && strcmp(start->first_name, first_name) == 0){
92
93         printf("Enter the Person's Number: ");
94         int num;
95         scanf("%d", &num);
96
97         if(num == start->num){
98             printf("\nThis record already exists.");
99             free(rec);
100             return start;
101         }
102
103         strncpy(rec->last_name, last_name, STRING_SIZE);
104         strncpy(rec->first_name, first_name, STRING_SIZE);
105         rec->num = num;
106
107         rec->next = start;
108         records[index] = rec;
109
110         printf("\nRecord added Successfully.");
111         return records[index];
112     }
113
114     rec = new_record(last_name, first_name);
115     struct record* temp = start;
116     while(temp->next != NULL){
117         temp = temp->next;
118     }
119     temp->next = rec;
120
121     return start;
122 }

```



```

170 int search_list(struct record* temp, char* last_name, char* first_name){
171     int found = 0;
172     while(temp){
173         if(strcmp(temp->last_name, last_name) == 0 && strcmp(temp->first_name, first_name) == 0){
174             printf("\n%d", temp->num);
175             found = 1;
176         }
177         temp = temp->next;
178     }
179     return found;
180 }

182 void display_record(char* last_name, char* first_name){
183     int index = find_hash(last_name);
184     int collisions = 1;
185     int found = 0;
186
187     if(LIST_LEN == num_terms){
188         int original_index = index;
189         while(strcmp(records[index]->last_name, last_name) != 0){
190             if(collisions == LIST_LEN){
191                 found = search_list(records[original_index], last_name, first_name);
192                 if(!found)
193                     printf("\nNo matching records found for this name.");
194                 return;
195             }
196             index = find_double_hash(last_name, collisions, index) % LIST_LEN;
197             ++collisions;
198         }
199         int secondary_found = search_list(records[index], last_name, first_name);
200         found = (found || secondary_found);
201     }
202
203     while(records[index] != NULL){
204         if(strcmp(records[index]->last_name, last_name) == 0){
205             found = search_list(records[index], last_name, first_name);
206             return;
207         }
208         index = find_double_hash(last_name, collisions, index) % LIST_LEN;
209         ++collisions;
210     }
211
212     printf("\nNo matching records found for this name.");
213 }

```

```

25 int main(){
26     while(1){
27         printf("\nPhone Directory\n");
28         printf("[1] Add Record\n");
29         printf("[2] Search Record\n");
30         printf("[3] Exit\n");
31
32         printf("Select one of the above options: ");
33         int option;
34         scanf("%d", &option);
35
36         char first_name[STRING_SIZE];
37         char last_name[STRING_SIZE];
38         printf("\n");
39         switch(option) {
40             case 1:
41                 printf("Enter the Person's First Name: ");
42                 scanf("%s", first_name);
43
44                 printf("Enter the Person's Last Name: ");
45                 scanf("%s", last_name);
46
47                 add_record(last_name, first_name);
48
49                 break;
50
51             case 2:
52                 printf("Enter Person's First Name to search for: ");
53                 scanf("%s", first_name);
54
55                 printf("Enter Person's Last Name to search for: ");
56                 scanf("%s", last_name);
57
58                 display_record(last_name, first_name);
59
60                 break;

```

```

51             case 2:
52                 printf("Enter Person's First Name to search for: ");
53                 scanf("%s", first_name);
54
55                 printf("Enter Person's Last Name to search for: ");
56                 scanf("%s", last_name);
57
58                 display_record(last_name, first_name);
59
60                 break;
61
62             case 3:
63                 exit(0);
64                 break;
65
66             default:
67                 printf("Enter a valid option.");
68
69         }
70         printf("\n");
71     }
72     return 0;
73 }

```

TEST CASES

TEST 1

```
Phone Directory
[1] Add Record
[2] Search Record
[3] Exit
Select one of the above options: 1

Enter the Person's First Name: Peeth
Enter the Person's Last Name: Chowdhary
Enter the Person's Number: 1234
```

Record added Successfully.

```
Phone Directory
[1] Add Record
[2] Search Record
[3] Exit
Select one of the above options: 1
```

```
Enter the Person's First Name: Peeth
Enter the Person's Last Name: Chowdhary
Enter the Person's Number: 5678
```

Record added Successfully.

```
Phone Directory
[1] Add Record
[2] Search Record
[3] Exit
Select one of the above options: 2
```

```
Enter Person's First Name to search for: Peeth
Enter Person's Last Name to search for: Chowdhary
```

```
5678
1234
```

```
Phone Directory
[1] Add Record
[2] Search Record
[3] Exit
Select one of the above options: 1

Enter the Person's First Name: Rohit
Enter the Person's Last Name: Deshpande
Enter the Person's Number: 3456
```

Record added Successfully.

```
Phone Directory
[1] Add Record
[2] Search Record
[3] Exit
Select one of the above options: 2
```

```
Enter Person's First Name to search for: Peeth
Enter Person's Last Name to search for: Chowdhary
```

No matching records found for this name.

```
Phone Directory
[1] Add Record
[2] Search Record
[3] Exit
Select one of the above options: 3
```

TEST 2

TEST CASES

```
Phone Directory
[1] Add Record
[2] Search Record
[3] Exit
Select one of the above options: 1

Enter the Person's First Name: Peeth
Enter the Person's Last Name: Chowdhary
Enter the Person's Number: 1234
```

Record added Successfully.

```
Phone Directory
[1] Add Record
[2] Search Record
[3] Exit
Select one of the above options: 1

Enter the Person's First Name: Rohit
Enter the Person's Last Name: Deshpande
Enter the Person's Number: 3456
```

Record added Successfully.

```
Phone Directory
[1] Add Record
[2] Search Record
[3] Exit
Select one of the above options: 1

Enter the Person's First Name: Eeshanya
Enter the Person's Last Name: Joshi
Enter the Person's Number: 6789
```

Record added Successfully.

Record added Successfully.

```
Phone Directory
[1] Add Record
[2] Search Record
[3] Exit
Select one of the above options: 1
```

```
Enter the Person's First Name: Sakshi
Enter the Person's Last Name: Raut
Enter the Person's Number: 8765
```

```
Phone Directory
[1] Add Record
[2] Search Record
[3] Exit
Select one of the above options: 2
```

```
Enter Person's First Name to search for: Sakshi
Enter Person's Last Name to search for: Raut
```

8765

```
Phone Directory
[1] Add Record
[2] Search Record
[3] Exit
Select one of the above options: 3
```

TEST 3

ANALYSIS OF RESULT

- The data structures used and the form of implementation ensures that regardless of the initial size of the hash table, efficiency is mostly maintained - unless the number of records hugely exceeds the size of the hash table.
- The updating of records already hashed by way of insertion of new phone numbers into the front of the linked lists ensures that the user is able to keep up to date with changing phone numbers.
- The code can be further extended by utilizing a priority queue as well as trees along with the double hashing that has been implemented. This may allow the user to further customization their experience as well as increase the execution speeds of functions.

Turnitin Originality Report

Processed on: 25-Oct-2023 22:53 IST
ID: 2207047950
Word Count: 555
Submitted: 1

dsia2code By Peeth Chowdhary

	Similarity by Source	
Similarity Index	5%	
	Internet Sources:	3%
	Publications:	1%
	Student Papers:	5%

2% match (Internet from 27-Jul-2021)
<https://cdmana.com/2021/04/20210428044005600e.html>

1% match (student papers from 15-Jan-2023)
[Submitted to University of Patras on 2023-01-15](#)

1% match (student papers from 15-Jan-2023)
[Submitted to University of Patras on 2023-01-15](#)

1% match (student papers from 25-Nov-2019)
[Submitted to Swinburne University of Technology on 2019-11-25](#)

```
#include <stdio.h> #include <stdlib.h> #include <string.h> #define LIST_LEN 3 #define STRING_SIZE 50 struct record* new_record(char* last_name, char* first_name); struct record* append_record(int index, char* last_name, char* first_name); unsigned int find_hash(char* s); unsigned int find_double_hash(char* s, int collisions, unsigned int hash); void add_record(char* last_name, char* first_name); void display_record(char* last_name, char* first_name); struct record{ int num; char first_name[STRING_SIZE]; char last_name[STRING_SIZE]; struct record* next; }; int num_terms = 0; struct record* records[LIST_LEN]; int main(){ while(1){ printf("\nPhone Directory\n"); printf("[1] Add Record\n"); printf("[2] Search Record\n"); printf("[3] Exit\n"); printf("Select one of the above options: "); int option; scanf("%d", &option); char first_name[STRING_SIZE]; char last_name[STRING_SIZE]; printf("\n"); switch(option) { case 1: printf("Enter the Person's First Name: "); scanf("%s", first_name); printf("Enter the Person's Last Name: "); scanf("%s", last_name); add_record(last_name, first_name); break; case 2: printf("Enter Person's First Name to search for: "); scanf("%s", first_name); printf("Enter Person's Last Name to search for: "); scanf("%s", last_name); display_record(last_name, first_name); break; case 3: exit(0); break; default: printf("Enter a valid option."); } printf("\n"); } return 0; } struct record* new_record(char* last_name, char* first_name){ struct record* rec = (struct record*)malloc(sizeof(struct record)); strncpy(rec->last_name, last_name, STRING_SIZE); strncpy(rec->first_name, first_name, STRING_SIZE); printf("Enter the Person's Number: "); scanf("%d", &rec->num); return rec; } struct record* append_record(int index, char* last_name, char* first_name){ struct record* rec = (struct record*)malloc(sizeof(struct record)); struct record* start = records[index]; if(strcmp(start->last_name, last_name) == 0 && strcmp(start->first_name, first_name) == 0){ printf("Enter the Person's Number: "); int num; scanf("%d", &num); if(num == start->num){ printf("\nThis record already exists."); free(rec); return start; } strncpy(rec->last_name, last_name, STRING_SIZE); strncpy(rec->first_name, first_name, STRING_SIZE); rec->num = num; rec->next = start; records[index] = rec; printf("\nRecord added Successfully."); return records[index]; } rec = new_record(last_name, first_name); struct record* temp = start; while(temp->next != NULL){ temp = temp->next; } temp->next = rec; return start; } unsigned int find_hash(char* s) { unsigned int hash = 0; while (*s) { hash = (hash * LIST_LEN) + *s; s++; } return hash % LIST_LEN; } unsigned int find_double_hash(char* s, int collisions, unsigned int hash) { return (hash + collisions * (1 + (hash % (LIST_LEN - 1))) + collisions) % LIST_LEN; } void add_record(char* last_name, char* first_name){ int index = find_hash(last_name); int collisions = 1; if(num_terms
```