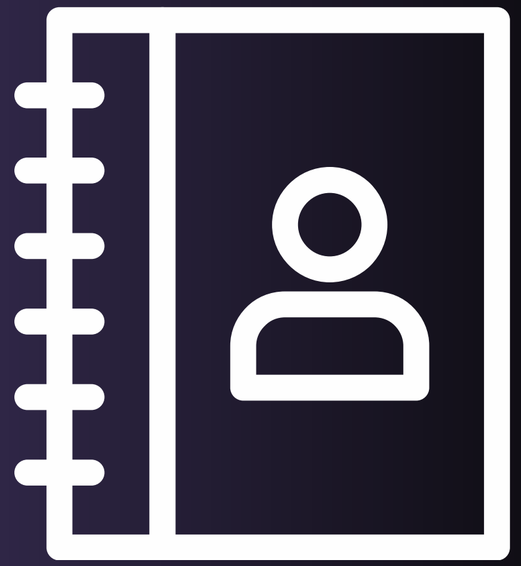
PHONE DIRECTORY

16010122034 - peeth chowdhary
16010122041 - rohit deshpande
16010122074 - eeshanya joshi

PROBLEM DEFINITION

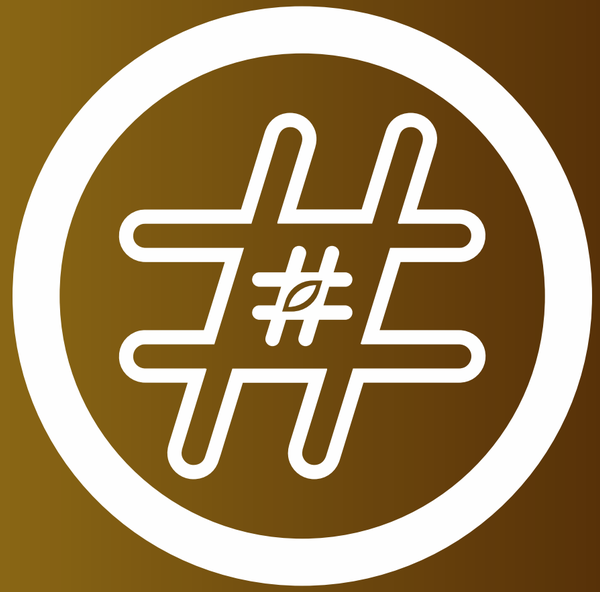
Creating a phone directory using hashing and linked lists in C. The user will be able to add multiple records of people's names their phone numbers. The user will also be able to search for a record in the given hash table and the extending linked lists.



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.

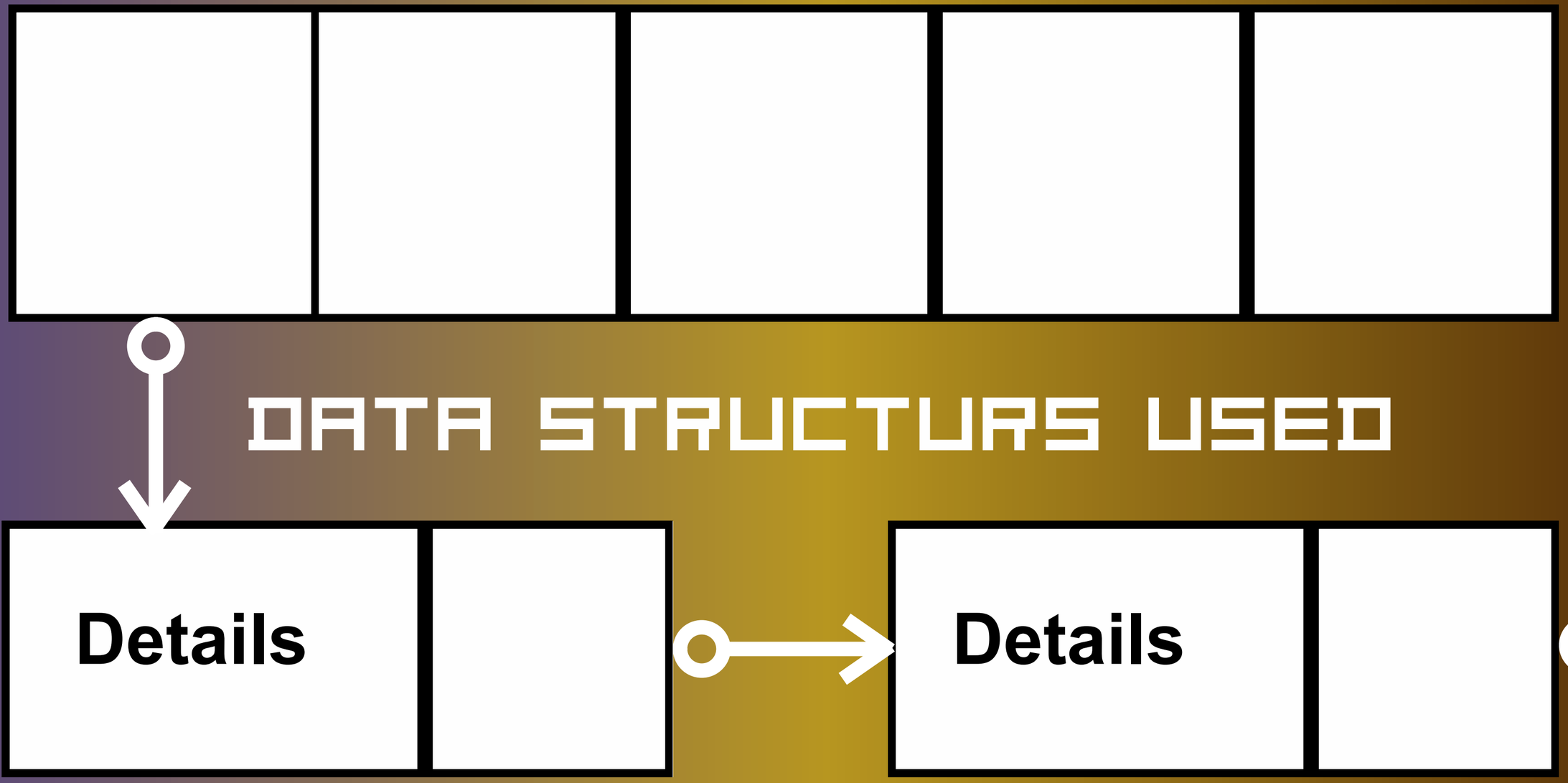


DATA STRUCTURES USED

- 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



Linked List starting from index 0

IMPLEMENTATION DETAILS

```c
#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];
```

```c
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;
}
```

```c
unsigned int find_hash(char* s) {
    unsigned int hash = 0;
    while (*s) {
        hash = (hash * LIST_LEN) + *s;
        s++;
    }
    return hash % LIST_LEN;
}
```

```c
unsigned int find_double_hash(char* s, int collisions, unsigned int hash) {
    return (hash + collisions * (1 + (hash % (LIST_LEN - 1))) + collisions) % LIST_LEN;
}
```

```c
void add_record(char* last_name, char* first_name){
    int index = find_hash(last_name);
    int collisions = 1;

    if(num_terms == LIST_LEN){
        int original_index = index;
        while(strcmp(records[index]->last_name, last_name) != 0){
            if(collisions == LIST_LEN){
                records[index] = append_record(original_index, last_name, first_name);
                return;
            }
            index = find_double_hash(last_name, collisions, index) % LIST_LEN;
            ++collisions;
        }
        records[index] = append_record(index, last_name, first_name);
        return;
    }

    while(records[index] != NULL){
        if(strcmp(records[index]->last_name, last_name) == 0){
            records[index] = append_record(index, last_name, first_name);
            return;
        }
        index = find_double_hash(last_name, collisions, index) % LIST_LEN;
        ++collisions;
    }

    struct record* rec = new_record(last_name, first_name);
    records[index] = rec;
    printf("\nRecord added Successfully.");
    num_terms++;
}
```

```c
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;
}
```

```c
int search_list(struct record* temp, char* last_name, char* first_name){
    int found = 0;
    while(temp){
        if(strcmp(temp->last_name, last_name) == 0 && strcmp(temp->first_name, first_name) == 0){
            printf("\n%d", temp->num);
            found = 1;
        }
        temp = temp->next;
    }
    return found;
}
```

```c
void display_record(char* last_name, char* first_name){
    int index = find_hash(last_name);
    int collisions = 1;
    int found = 0;

    if(LIST_LEN == num_terms){
        int original_index = index;
        while(strcmp(records[index]->last_name, last_name) != 0){
            if(collisions == LIST_LEN){
                found = search_list(records[original_index], last_name, first_name);
                if(!found)
                    printf("\nNo matching records found for this name.");
                return;
            }
            index = find_double_hash(last_name, collisions, index) % LIST_LEN;
            ++collisions;
        }
        int secondary_found = search_list(records[index], last_name, first_name);
        found = (found || secondary_found);
    }

    while(records[index] != NULL){
        if(strcmp(records[index]->last_name, last_name) == 0){
            found = search_list(records[index], last_name, first_name);
            return;
        }
        index = find_double_hash(last_name, collisions, index) % LIST_LEN;
        ++collisions;
    }

    printf("\nNo matching records found for this name.");
}
```

```c
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 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;
}
```

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
```

TEST 2

```
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 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 Index	Similarity by Source	
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