DATA STRUCTURES LABORATORY

Course Code: 1BCSL306

VTU 2025 Scheme – Semester 3

PART – B

Open-Ended Experiment

EXPERIMENT – 11

Employee Record Management

Using Hash Table Indexing

Simple Code • Fast Search • Exam Friendly

Your VTU Study Partner

Experiment 11

1. Aim

Aim

To develop a C program to store and manage employee records using a suitable indexing mechanism and perform insertion, search, deletion and department-wise display.

2. VTU Prescribed Experiment

VTU Prescribed Experiment

Develop a C program to store employee records having fields:

- Employee Number
- Name
- Designation
- Department

using a suitable indexing mechanism.
The program shall support record management operations such as:

- Adding a new employee.
- Search for employee details based on employee number.
- Remove employee record based on employee number.
- Display employee records based on department.

Demonstrate the correctness of the implementation using appropriate test cases.

3. Suitable Indexing Mechanism

For this experiment, we use:

Hash Table

The Employee Number is used as the key.

Hashing provides fast:

Insert

$$\boxed{Search}$$

$$\boxed{Delete}$$

4. Simple Theory

A hash table stores records using an index calculated from a key.

Here:

$$\boxed{Key = Employee\ Number}$$

A simple hash function is:

$$\boxed{Index = EmployeeNo\%SIZE}$$

If the calculated position is already occupied, use:

$$\boxed{Linear\ Probing}$$

5. Employee Record

Each employee contains:

$$\boxed{EmpNo + Name + Designation + Department}$$

C structure:

```
struct Employee
{
    int empNo;
    char name[30];
    char designation[30];
    char department[20];
    int used;
};
```

6. Hash Function

For table size 10:

$$\boxed{hash = empNo\%10}$$

Example:

$$EmployeeNo = 105$$

then:

$$105\%10 = 5$$

Therefore the record is stored at:

$$\boxed{Index\ 5}$$

7. Collision Handling

If two Employee Numbers produce the same index, a collision occurs.
Example:

$$105\%10 = 5$$

and

$$115\%10 = 5$$

Both generate index 5.
Use Linear Probing:

$$\boxed{index = (index + 1)\%SIZE}$$

until an empty position is found.

8. Algorithm

8.1. Add Employee

1. Read Employee Number.
2. Calculate hash index.
3. Check whether the position is free.
4. If occupied, perform linear probing.
5. Read Name, Designation and Department.
6. Store the employee record.

8.2. Search Employee

1. Read Employee Number.
2. Search the hash table.
3. Compare employee numbers.
4. If found, display complete employee details.
5. Otherwise display "Employee Not Found".

8.3. Remove Employee

1. Read Employee Number.
2. Search for the employee record.
3. If found, mark that record as unused.
4. Display successful deletion message.

8.4. Display by Department

1. Read Department.
2. Scan all hash table entries.
3. Compare department name.
4. Display matching employee records.

Easy Exam Memory

Experiment 11:

$$\boxed{Employee \rightarrow Hash \rightarrow Add \rightarrow Search \rightarrow Delete \rightarrow Department}$$

Employee fields:

$$\boxed{EmpNo,\ Name,\ Designation,\ Department}$$

9. C Program – Easy Student Version

```c
#include<stdio.h>
#include<string.h>

#define SIZE 10

struct Employee
{
    int empNo;
    char name[30];
    char designation[30];
    char department[20];
    int used;
};

struct Employee table[SIZE];

int hash(int empNo)
{
    return empNo%SIZE;
}

void addEmployee()
{
    int index,start,empNo;

    printf("Enter Employee Number: ");
    scanf("%d",&empNo);

    index=hash(empNo);
    start=index;

    while(table[index].used)
    {
        index=(index+1)%SIZE;

        if(index==start)
        {
            printf("Table Full\n");
            return;
        }
    }

    table[index].empNo=empNo;

    printf("Enter Name: ");
    scanf("%s",table[index].name);

    printf("Enter Designation: ");
    scanf("%s",table[index].designation);

```

```
    printf("Enter Department: ");
    scanf("%s",table[index].department);

    table[index].used=1;

    printf("Employee Added Successfully\n");
}

void searchEmployee()
{
    int empNo,i;

    printf("Enter Employee Number: ");
    scanf("%d",&empNo);

    for(i=0;i<SIZE;i++)
    {
        if(table[i].used &&
           table[i].empNo==empNo)
        {
            printf("\nEmployee Number: %d",
                   table[i].empNo);

            printf("\nName: %s",
                   table[i].name);

            printf("\nDesignation: %s",
                   table[i].designation);

            printf("\nDepartment: %s\n",
                   table[i].department);

            return;
        }
    }

    printf("Employee Not Found\n");
}

void removeEmployee()
{
    int empNo,i;

    printf("Enter Employee Number: ");
    scanf("%d",&empNo);

    for(i=0;i<SIZE;i++)
    {
        if(table[i].used &&
           table[i].empNo==empNo)
        {
            table[i].used=0;

```

```
            printf("Employee Removed Successfully\n");
            return;
        }
    }

    printf("Employee Not Found\n");
}

void displayDepartment()
{
    char dept[20];
    int i,found=0;

    printf("Enter Department: ");
    scanf("%s",dept);

    for(i=0;i<SIZE;i++)
    {
        if(table[i].used &&
           strcmp(table[i].department,dept)==0)
        {
            printf("\n%d  %s  %s  %s",
                   table[i].empNo,
                   table[i].name,
                   table[i].designation,
                   table[i].department);

            found=1;
        }
    }

    if(found==0)
        printf("No Employees Found\n");

    printf("\n");
}

void displayAll()
{
    int i;

    for(i=0;i<SIZE;i++)
    {
        if(table[i].used)
        {
            printf("\n%d  %s  %s  %s",
                   table[i].empNo,
                   table[i].name,
                   table[i].designation,
                   table[i].department);
        }
    }

```

```
    printf("\n");
}

int main()
{
    int ch;

    do
    {
        printf("\n1.Add Employee");
        printf("\n2.Search Employee");
        printf("\n3.Remove Employee");
        printf("\n4.Display by Department");
        printf("\n5.Display All");
        printf("\n6.Exit");

        printf("\nEnter Choice: ");
        scanf("%d",&ch);

        switch(ch)
        {
            case 1:
                addEmployee();
                break;

            case 2:
                searchEmployee();
                break;

            case 3:
                removeEmployee();
                break;

            case 4:
                displayDepartment();
                break;

            case 5:
                displayAll();
                break;

            case 6:
                printf("Program Ended\n");
                break;

            default:
                printf("Invalid Choice\n");
        }

    }while(ch!=6);

    return 0;
}
```

10. Simple Program Explanation

10.1. hash()

The function:

```
return empNo%SIZE;
```

converts Employee Number into a hash-table index.

10.2. addEmployee()

Calculate index:

```
index=hash(empNo);
```

If the position is already occupied:

```
index=(index+1)%SIZE;
```

This is called:

Linear Probing

10.3. searchEmployee()

The important condition is:

```
table[i].empNo==empNo
```

If true, the employee details are displayed.

10.4. removeEmployee()

When the employee is found:

```
table[i].used=0;
```

The record becomes inactive.

10.5. displayDepartment()

Department names are compared using:

```
strcmp(table[i].department,dept)==0
```

Only matching employees are displayed.

Easy Exam Memory

Important functions:

$$\boxed{hash()}$$

$$\boxed{addEmployee()}$$

$$\boxed{searchEmployee()}$$

$$\boxed{removeEmployee()}$$

$$\boxed{displayDepartment()}$$

Easy order:

$$\boxed{Hash \rightarrow Add \rightarrow Search \rightarrow Delete \rightarrow Display}$$

11. Sample Input and Output

```

1.Add Employee
2.Search Employee
3.Remove Employee
4.Display by Department
5.Display All
6.Exit

Enter Choice: 1

Enter Employee Number: 101
Enter Name: Ravi
Enter Designation: Developer
Enter Department: CSE

Employee Added Successfully


Enter Choice: 1

Enter Employee Number: 102
Enter Name: Anu
Enter Designation: Manager
Enter Department: HR

Employee Added Successfully


Enter Choice: 1

Enter Employee Number: 111
Enter Name: Kiran
Enter Designation: Tester
Enter Department: CSE

Employee Added Successfully
```


12. Search Example

Input:

```

Enter Choice: 2
Enter Employee Number: 101
```

Output:

```

Employee Number: 101
```

```
Name: Ravi
Designation: Developer
Department: CSE
```

13. Department-wise Display

Input:

```

Enter Choice: 4
Enter Department: CSE
```

Output:

```

101  Ravi   Developer  CSE
111  Kiran  Tester     CSE
```

14. Remove Employee Example

Input:

```

Enter Choice: 3
Enter Employee Number: 102
```

Output:

```

Employee Removed Successfully
```

15. Appropriate Test Cases

Test	Operation	Expected Result
1	Add Employee 101	Employee is inserted successfully.
2	Search Employee 101	Employee details are displayed.
3	Search Employee 999	Employee Not Found.
4	Remove Employee 101	Employee is removed.
5	Display CSE Department	Only CSE employees are displayed.
6	Insert colliding keys 101 and 111	Linear probing stores records separately.

16. Performance

For a good hash table:

$$\boxed{Insertion \approx O(1)}$$

$$\boxed{Search \approx O(1)}$$

$$\boxed{Deletion \approx O(1)}$$

Department-wise display requires scanning the table:

$$\boxed{O(n)}$$

17. Result

Result

Thus, a C program to manage Employee Records using a Hash Table as an indexing mechanism was successfully implemented.
The program supports:

- Adding new employee records.
- Searching employee details using employee number.
- Removing employee records using employee number.
- Displaying employee records based on department.
- Demonstrating the implementation using suitable test cases.

18. Important Viva Questions

Important Viva Questions

1. What is indexing?
Indexing is a technique used to locate records quickly using a key.

2. Which indexing mechanism is used in this program?
Hash Table.

3. What is the key used in this experiment?
Employee Number.

4. What fields are stored in an employee record?
Employee Number, Name, Designation and Department.

5. What is the hash function used?

$$\boxed{empNo\%SIZE}$$

6. What is hashing?
Hashing maps a key to an index in a hash table.

7. What is collision?
Collision occurs when two different keys produce the same index.

8. Which collision handling method is used?
Linear Probing.

9. What is the Linear Probing formula?

$$\boxed{(index+1)\%SIZE}$$

10. Which function compares department names?

`strcmp()`

11. Which header file provides strcmp()?

`string.h`

12. What does used=1 indicate?
The table location contains an active employee record.

13. What does used=0 indicate?
The location is considered unused.

14. What is average hash search complexity?

$$\boxed{O(1)}$$

15. Why is hashing useful?
It provides fast insertion, searching and deletion.

19. 1-Minute Exam Revision

Easy Exam Memory

Fields

$$\boxed{EmpNo + Name + Designation + Department}$$

Index

$$\boxed{empNo\%SIZE}$$

Collision

$$\boxed{Linear\ Probing}$$

Add

$$\boxed{Hash \rightarrow Empty\ Position \rightarrow Store}$$

Search

$$\boxed{Employee\ Number}$$

Delete

$$\boxed{used = 0}$$

Department Search

$$\boxed{strcmp()}$$

20. Easy Code Memory Map

Remember

Remember the code as:

$$\boxed{Employee\ Structure}$$

$$\downarrow$$

$$\boxed{hash()}$$

$$\downarrow$$

$$\boxed{addEmployee()}$$

$$\downarrow$$

$$\boxed{searchEmployee()}$$

$$\downarrow$$

$$\boxed{removeEmployee()}$$

$$\downarrow$$

$$\boxed{displayDepartment()}$$

Most important formula:

$$\boxed{Index = EmpNo\%SIZE}$$

SEARCH CREATORS

1BCSL306 – Data Structures Laboratory

PART – B

EXPERIMENT 11
COMPLETED

Employee Record Management

Using Hash Table Indexing

Add → Search → Remove → Department → Test

Your VTU Study Partner