Task 12

Task 12a: To write a C program to implement the concept of Indexing

```
#include<stdio.h>
#include<string.h>
int n;
void main()
 {
   int b[20], b1[20], i, j, blocks[20][20], sz[20];
   char F[20][20], S[20], ch;
   int sb[20], eb[20], x, block_found;

   // Input the number of files
   printf("\n Enter number of Files: ");
   scanf("%d", &n);

   // Input file names, sizes, and block sizes for each file
   for(i = 0; i < n; i++) {
      printf("\n Enter file %d name: ", i + 1);
      scanf("%s", F[i]);

      printf("\n Enter file %d size (in KB): ", i + 1);
      scanf("%d", &sz[i]);

      printf("\n Enter block size of file %d (in bytes): ", i + 1);
      scanf("%d", &b[i]);
   }
```

```
// Calculating the number of blocks needed for each file and accepting block numbers
for(i = 0; i < n; i++) {
    b1[i] = (sz[i] * 1024) / b[i];  // Calculate the number of blocks
    printf("\n Enter Starting block of file %d: ", i + 1);
    scanf("%d", &sb[i]);

    printf("\n Enter Ending block of file %d: ", i + 1);
    scanf("%d", &eb[i]);

    printf("\n Enter blocks for file %d:\n", i + 1);
    for(j = 0; j < b1[i] - 2;) {  // Loop through to enter block numbers
        printf("\n Enter the %d block: ", j + 1);
        scanf("%d", &x);

        // Validate the block number
        if(x >= sb[i] && x <= eb[i]) {
            block_found = 0;
            // Check if block is already allocated
            for(int k = 0; k < j; k++) {
                if(blocks[i][k] == x) {
                    block_found = 1;
                    printf("\n Duplicate block entered.\n");
                    break;
                }
```

```
        }
        if(block_found == 0) {
            blocks[i][j] = x;
            j++;
        }
      } else {
        printf("\n Invalid block, it must be within the range %d to %d.\n", sb[i], eb[i]);
      }
   }
}

// Loop to display file information
do {
   printf("\nEnter the Filename: ");
   scanf("%s", S);

   for(i = 0; i < n; i++) {
      if(strcmp(F[i], S) == 0) {  // Search for the entered filename
         printf("\nFname\tFsize\tBsize\tNblocks\tBlocks\n");
         printf("\n \n");
         printf("\n%s\t%d\t%d\t%d\t", F[i], sz[i], b[i], b1[i]);
         printf("%d->", sb[i]);
         for(j = 0; j < b1[i] - 2; j++) {
            printf("%d->", blocks[i][j]);
         }
```

```
            printf("%d->", eb[i]);
          }
        }

        printf("\n \n");
        printf("\nDo you want to continue (Y/n): ");
        scanf(" %c", &ch);  // Corrected to accept single char input
    } while(ch != 'n' && ch != 'N');  // Checking both lowercase and uppercase 'n'
}
```

OUTPUT:

```
Enter number of Files: 2
 Enter file 1 name: FILE1
 Enter file 1 size (in KB): 1
 Enter block size of file 1 (in bytes): 512
 Enter file 2 name: FILE2
 Enter file 2 size (in KB): 1
 Enter block size of file 2 (in bytes): 512
 Enter Starting block of file 1: 2
 Enter Ending block of file 1: 5
 Enter blocks for file 1:
 Enter Starting block of file 2: 6
 Enter Ending block of file 2: 10
 Enter blocks for file 2:
Enter the Filename: FILE1
Fname   Fsize   Bsize   Nblocks Blocks
```

```
FILE1   1      512     2      2->5->

Do you want to continue (Y/n): y

Enter the Filename: FILE2

Fname   Fsize   Bsize   Nblocks Blocks

FILE2   1      512     2      6->10->

Do you want to continue (Y/n): n
```

Task 12b: To write a C program to implement the concept of Hashing.

Program:

```
#include <stdio.h>
#include <limits.h>  // For INT_MIN and INT_MAX

// Function to insert an element using linear probing
void insert(int ary[], int hFn, int size) {
   int element, pos, n = 0;

   printf("Enter key element to insert: ");
   scanf("%d", &element);

   // Calculate initial hash index
   pos = element % hFn;

   // Linear probing to handle collision
   while (ary[pos] != INT_MIN && ary[pos] != INT_MAX) {
      pos = (pos + 1) % hFn;
      n++;
      if (n == size) break;  // Stop if table is full
   }

   // Insert if space is available
   if (n == size) {
      printf("Hash table is full. Cannot insert the element.\n\n");
   } else {
      ary[pos] = element;
   }
}

// Function to delete an element
void delet(int ary[], int hFn, int size) {
```

```
    int element, n = 0, pos;

    printf("Enter element to delete: ");
    scanf("%d", &element);

    // Calculate initial hash index
    pos = element % hFn;

    // Search for the element with linear probing
    while (n++ != size) {
        if (ary[pos] == INT_MIN) {
            // Empty cell means element not found
            printf("Element not found in hash table.\n");
            break;
        } else if (ary[pos] == element) {
            // Found the element, mark it as deleted
            ary[pos] = INT_MAX;
            printf("Element deleted.\n\n");
            break;
        } else {
            // Move to next cell
            pos = (pos + 1) % hFn;
        }
    }

    if (--n == size) {
        printf("Element not found in hash table.\n");
    }
}

// Function to search for an element
void search(int ary[], int hFn, int size) {
    int element, pos, n = 0;

    printf("Enter element you want to search: ");
    scanf("%d", &element);

    // Calculate initial hash index
    pos = element % hFn;

    // Linear probing to find the element
    while (n++ != size) {
        if (ary[pos] == element) {
            printf("Element found at index %d\n", pos);
            break;
        } else if (ary[pos] == INT_MIN) {
            // Empty cell means element not found
            printf("Element not found in hash table.\n");
```

```
            break;
        } else {
            pos = (pos + 1) % hFn;
        }
    }

    if (--n == size) {
        printf("Element not found in hash table.\n");
    }
}

// Function to display hash table
void display(int ary[], int size) {
    int i;

    printf("Index\tValue\n");
    for (i = 0; i < size; i++) {
        if (ary[i] == INT_MIN || ary[i] == INT_MAX)
            printf("%d\t---\n", i);  // Empty or deleted
        else
            printf("%d\t%d\n", i, ary[i]);  // Actual value
    }
}

// Main function
int main() {
    int size, hFn, i, choice;

    // Input size of hash table
    printf("Enter size of hash table: ");
    scanf("%d", &size);

    int ary[size];  // Declare hash table array

    // Input hash function modulus
    printf("Enter hash function modulus (e.g., 10 for mod 10): ");
    scanf("%d", &hFn);

    // Initialize hash table with INT_MIN (empty)
    for (i = 0; i < size; i++) {
        ary[i] = INT_MIN;
    }

    // Menu-driven interface
    do {
        printf("\nEnter your choice:\n");
        printf("1 -> Insert\n2 -> Delete\n3 -> Display\n4 -> Search\n0 -> Exit\n");
        scanf("%d", &choice);
```

```
        switch (choice) {
            case 1: insert(ary, hFn, size); break;
            case 2: delet(ary, hFn, size); break;
            case 3: display(ary, size); break;
            case 4: search(ary, hFn, size); break;
            case 0: printf("Exiting...\n"); break;
            default: printf("Enter correct choice!\n");
        }
    } while (choice != 0);

    return 0;
}
```

Output:

```
Enter size of hash table: 5
Enter hash function modulus (e.g., 10 for mod 10): 5

Enter your choice:
1 -> Insert
2 -> Delete
3 -> Display
4 -> Search
0 -> Exit
1
Enter key element to insert: 10

Enter your choice:
1 -> Insert
2 -> Delete
3 -> Display
4 -> Search
0 -> Exit
1
Enter key element to insert: 20

Enter your choice:
1 -> Insert
2 -> Delete
3 -> Display
4 -> Search
0 -> Exit
1
Enter key element to insert: 30

Enter your choice:
1 -> Insert
2 -> Delete
```

```
3 -> Display
4 -> Search
0 -> Exit
3
Index   Value
0       10
1       20
2       30
3       ---
4       ---

Enter your choice:
1 -> Insert
2 -> Delete
3 -> Display
4 -> Search
0 -> Exit
3
Index   Value
0       10
1       20
2       30
3       ---
4       ---

Enter your choice:
1 -> Insert
2 -> Delete
3 -> Display
4 -> Search
0 -> Exit
4
Enter element you want to search: 20
Element found at index 1

Enter your choice:
1 -> Insert
2 -> Delete
3 -> Display
4 -> Search
0 -> Exit
```