DATA STRUCTURES LABORATORY

18CSL38

ATRIA INSTITUTE OF TECHNOLOGY
DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
Bengaluru - 560024
2020

DATA STRUCTURES LABORATORY
(Effective from the academic year 2018 -2019)
SEMESTER – III

Course Code	18CSL38	**CIE Marks**	40
Number of Contact Hours/Week	0:2:2	**SEE Marks**	60
Total Number of Lab Contact Hours	36	**Exam Hours**	3 Hrs

Credits – 2

Course Learning Objectives: This course (18CSL38) will enable students to:

This laboratory course enable students to get practical experience in design, develop, implement, analyze and evaluation/testing of

- Asymptotic performance of algorithms.
- Linear data structures and their applications such as stacks, queues and lists
- Non-Linear data structures and their applications such as trees and graphs
- Sorting and searching algorithms

Descriptions (if any):

- Implement all the programs in 'C / C++' Programming Language and Linux / Windows as OS.

Programs List:

1.	Design, Develop and Implement a menu driven Program in C for the following array operations. a. Creating an array of N Integer Elements b. Display of array Elements with Suitable Headings c. Inserting an Element (ELEM) at a given valid Position (POS) d. Deleting an Element at a given valid Position (POS) e. Exit. Support the program with functions for each of the above operations.
2.	Design, Develop and Implement a Program in C for the following operations on Strings. a. Read a main String (STR), a Pattern String (PAT) and a Replace String (REP) b. Perform Pattern Matching Operation: Find and Replace all occurrences of PAT in STR with REP if PAT exists in STR. Report suitable messages in case PAT does not exist in STR Support the program with functions for each of the above operations. Don't use Built-in functions.
3.	Design, Develop and Implement a menu driven Program in C for the following operations on STACK of Integers (Array Implementation of Stack with maximum size MAX) a. Push an Element on to Stack b. Pop an Element from Stack c. Demonstrate how Stack can be used to check Palindrome d. Demonstrate Overflow and Underflow situations on Stack e. Display the status of Stack f. Exit Support the program with appropriate functions for each of the above operations
4.	Design, Develop and Implement a Program in C for converting an Infix Expression to Postfix Expression. Program should support for both parenthesized and free parenthesized expressions with the operators: +, -, *, /, % (Remainder), ^ (Power) and alphanumeric operands.

5.	Design, Develop and Implement a Program in C for the following Stack Applications a. Evaluation of Suffix expression with single digit operands and operators: +, -, *, /, %, ^ b. Solving Tower of Hanoi problem with n disks
6.	Design, Develop and Implement a menu driven Program in C for the following operations on Circular QUEUE of Characters (Array Implementation of Queue with maximum size MAX) a. Insert an Element on to Circular QUEUE b. Delete an Element from Circular QUEUE c. Demonstrate Overflow and Underflow situations on Circular QUEUE d. Display the status of Circular QUEUE e. Exit Support the program with appropriate functions for each of the above operations
7.	Design, Develop and Implement a menu driven Program in C for the following operations on Singly Linked List (SLL) of Student Data with the fields: *USN, Name, Programme, Sem, PhNo* a. Create a SLL of N Students Data by using *front insertion.* b. Display the status of SLL and count the number of nodes in it c. Perform Insertion / Deletion at End of SLL d. Perform Insertion / Deletion at Front of SLL(Demonstration of stack) e. Exit
8.	Design, Develop and Implement a menu driven Program in C for the following operations on Doubly Linked List (DLL) of Employee Data with the fields: *SSN, Name, Dept, Designation, Sal, PhNo* a. Create a DLL of N Employees Data by using *end insertion.* b. Display the status of DLL and count the number of nodes in it c. Perform Insertion and Deletion at End of DLL d. Perform Insertion and Deletion at Front of DLL e. Demonstrate how this DLL can be used as Double Ended Queue. f. Exit
9.	Design, Develop and Implement a Program in C for the following operationson Singly Circular Linked List (SCLL) with header nodes a. Represent and Evaluate a Polynomial $P(x,y,z) = 6x^2y^2z-4yz^5+3x^3yz+2xy^5z-2xyz^3$ b. Find the sum of two polynomials POLY1(x,y,z) and POLY2(x,y,z) and store the result in POLYSUM(x,y,z) Support the program with appropriate functions for each of the above operations
10.	Design, Develop and Implement a menu driven Program in C for the following operations on Binary Search Tree (BST) of Integers . a. Create a BST of N Integers: 6, 9, 5, 2, 8, 15, 24, 14, 7, 8, 5, 2 b. Traverse the BST in Inorder, Preorder and Post Order c. Search the BST for a given element (KEY) and report the appropriate message d. Exit
11.	Design, Develop and Implement a Program in C for the following operations on Graph(G) of Cities a. Create a Graph of N cities using Adjacency Matrix. b. Print all the nodes reachable from a given starting node in a digraph using DFS/BFS

	method
12.	Given a File of N employee records with a set K of Keys (4-digit) which uniquely determine the records in file F. Assume that file F is maintained in memory by a Hash Table (HT) of m memory locations with L as the set of memory addresses (2-digit) of locations in HT. Let the keys in K and addresses in L are Integers. Design and develop a Program in C that uses Hash function H: K $\rightarrow$L as H(K)=K mod m (remainder method), and implement hashing technique to map a given key K to the address space L. Resolve the collision (if any) using linear probing.

Laboratory Outcomes: The student should be able to:

- Analyze and Compare various linear and non-linear data structures
- Code, debug and demonstrate the working nature of different types of data structures and their applications
- Implement, analyze and evaluate the searching and sorting algorithms
- Choose the appropriate data structure for solving real world problems

Conduct of Practical Examination:

- Experiment distribution
  - For laboratories having only one part: Students are allowed to pick one experiment from the lot with equal opportunity.
  - For laboratories having PART A and PART B: Students are allowed to pick one experiment from PART A and one experiment from PART B, with equal opportunity.
- Change of experiment is allowed only once and marks allotted for procedure to be made zero of the changed part only.
- Marks Distribution *(Courseed to change in accoradance with university regulations)*
  c) For laboratories having only one part – Procedure + Execution + Viva-Voce: 15+70+15 = 100 Marks
  d) For laboratories having PART A and PART B
    i. Part A – Procedure + Execution + Viva = 6 + 28 + 6 = 40 Marks
    ii. Part B – Procedure + Execution + Viva = 9 + 42 + 9 = 60 Marks

Contents

1 Array

Design, Develop and Implement a menu driven Program in C for the following array operations.

```
a. Creating an array of N Integer Elements
b. Display of array Elements with Suitable Headings
c. Inserting an Element (ELEM) at a given valid Position (POS)
d. Deleting an Element at a given valid Position (POS)
e. Exit.
```

Support the program with functions for each of the above operations.

1.1 Test Cases.

```
---------- Menu ----------
1. Create
2. Display
3. Insert
4. Delete
5. Exit
Enter your choice: 1
```

1. Create array with data elements 2 4 6 8 10 and display
2. Insert 12 at pos=0 and display
3. Delete item at pos=5 and display
4. Exit

1.2 Execution steps

1. gcc -g 1.c
2. ./a.out

1.3 Program

```c
/* 1.c Array */

#include <stdio.h>
#include <stdlib.h>

int a[20];
int n = 0;

/*
1. Prompt for array size (n).
2. Read n array elements
*/

void create()
{
  printf("Enter the size of the array: ");
  scanf("%d", &n);

  printf("Enter the elements for the array:\n");
  for(int i=0; i<n; i++)
    {
      scanf("%d",&a[i]);
    }
}

/*
1. Print proper heading
2. print n array elements
*/

void display()
{
  printf("The array elements are:\n");
  for(int i=0; i<n; i++)
    {
      printf("%d\n",a[i]);
    }
}

/*
1. Prompt for pos
2. Read pos

3. Prompt for element value
4. Read the element
```

```
5. Shift right the elements n-1, n-2, .., pos
6. Insert a[pos] = value

7. Increment the array size n
*/

void insert()
{
  int pos, value;

  printf("Enter the index position for the new element: ");
  scanf("%d", &pos);

  printf("Enter the element to be inserted : ");
  scanf("%d", &value);

  for (int i=n-1; i>=pos; i--)
    {
      a[i+1]=a[i];
    }
  a[pos] = value;
  n=n+1;
}

/*
1. Prompt for pos
2. read pos
3. shift left the elements pos+1 ,..., n-1
4. Decrement the array size n
*/

void delete()
{
  int pos, value;

  printf("Enter the index position of the element to be deleted: ");
  scanf("%d", &pos);

  value = a[pos];
  for(int i=pos+1; i<n; i++)
    {
      a[i-1]=a[i];
    }
  n = n-1;
  printf("The deleted element is = %d",value);
}

/*
1. Define Menu type
```

```
2. Declare menu as an array variable of type Menu and  initialize.

3. Print menu header
4. Display menu names

5. Prompt for choice
6. Read choice
7. Invoke the function corresponding to the choice menu[choice].func()

8. Repeat steps 3, 4, 5,6,7 for ever
 */

int main( )
{
  int choice;

  struct MENU
  {
    char *name; void (*func)();
  } menu[] = {
    {"Done",    exit   },
    {"Create",  create },
    {"Insert",  insert },
    {"Delete",  delete },
    {"Display", display}
  };

  setvbuf (stdout, (char *) NULL, _IONBF, 0);
  for(;;)
    {
      printf("\n-------Array Menu-------\n");
      for(int i=1; i < 5; i++)
        {
          printf("%d. %s\n", i, menu[i].name);
        }
      printf("--------------------------------\n");

      printf("Enter your choice: ");
      scanf("%d", &choice);

      menu[choice].func();
    }
}
```

2 Strings

Design, Develop and Implement a Program in C for the following operations on Strings.

a. Read a main String (STR), a Pattern String (PAT) and a Replace String (REP)
b. Perform Pattern Matching Operation: Find and Replace all occurrences of PAT in STR with REP if PAT exists in STR. Report suitable messages in case PAT does not exist in STR

Support the program with functions for each of the above operations. Don't use Built-in functions.

2.1 Test cases

```
Enter t: ababbaabaa
Enter p: aab
Enter r: AAB
pattern occur at the postion 5
t: ababbAABaa
```

2.2 Execution steps

1. gcc -g 2.c
2. ./a.out

2.3 Program

```c
/* 2 Strings */
#include <stdio.h>
#include <string.h>

/*
1. r >= p
2. Shift right t from position s by d chars
3. Terminate the shifted string by null
*/
int shift_r(char t[], int s, int d)
{
  int i;
  int n = strlen(t);

  for (i=n-1; i >= s+d; i--)
    {
      t[i+d] = t[i];
    }
  t[n+d] = '\0';
}

/*
1. r < p
2. Shift left t from position s by d chars
2. Terminate the shifted string by null
*/
int shift_l(char t[], int s, int d)
{
  int i;
  int n = strlen(t);

  for (i=s+d; i < n; i++)
    {
      t[i-d] = t[i];
    }
  t[n-d] = '\0';
}
/*
1. Inputs are text(t), pattern(p), replace(r)
2. Find the position(s) of the pattern(p) in the text(t)
3. Delete pattern(p) at the position (s)
4. Insert the replaceable string(r) at the position s
*/

void pattern(char t[], char p[], char r[])
{
```

```
  int i;

  int n = strlen(t);
  int m = strlen(p);
  int l = strlen(r);
  int d = l - m;

  for(int s = 0; s < n-m+1; s++)
    {
      if (strncmp(t+s, p, m) == 0)
        {
          // make space for string r
          d < 0 ? shift_l(t, s, -d) : shift_r(t, s, d);

          // embed r into t
          for (int i=0; i < l; i++)
            {
              t[s++] = r[i];
            }
          s--;
        }
    }
}

int main() {
  char t[50], p[10], r[10];

  printf("Enter t: ");
  gets(t);

  printf("Enter p: ");
  gets(p);

  printf("Enter r: ");
  gets(r);

  pattern(t, p, r);

  printf("t: %s\n", t);
}
```

3 Stack

Design, Develop and Implement a menu driven Program in C for the following operations on STACK of Integers (Array Implementation of Stack with maximum size MAX)

```
a. Push an Element on to Stack
b. Pop an Element from Stack
c. Demonstrate how Stack can be used to check Palindrome
d. Demonstrate Overflow and Underflow situations on Stack
e. Display the status of Stack
f. Exit
```

Support the program with appropriate functions for each of the above operations

3.1 Test cases

1. pop() results in Underflow
2. push() 5 elements, for 6th element the result is Overflow
3. push(10)
4. palindrome 252
5. Display stack with 10 (contents before palindrome)

3.2 Execution steps

1. gcc -g 3.c
2. ./a.out

3.3 Program

```c
/* 3.c Stack */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define SIZE 10

int stack[SIZE];
int top = -1;

/*
1. Input is item
2. Check stackfull condition
3. Increment top
4. Store item in the stack
 */
void push(int item)
{
  if (top != SIZE-1)
    {
      stack[++top]=item;
    }
  else printf("Stack Overflow\n");
}

/*
1. Output is item
2. Check stack empty condition
3. Remove item in the stack
4. Decrement the top
 */
int pop()
{
  if (top != -1)
    {
      return(stack[top--]);
    }
  else printf("Stack Underflow\n");
}

/*
1. Check stack empty condition
2. Print all the items on the stack
*/
void display()
{
```

```
  if (top == -1)
    {
      printf("Stack EMPTY\n");
      return;
    }

  printf("The Stack status is \n");
  for(int i=top; i >= 0; i--)
    {
      printf("%d\n", stack[i]);
    }
}

/*
1. Read a number
2. Convert number to string 's'
3. Push the string chars on to the stack
4. Pop the stack chars into 'r'
5. Compare 's' and 'r' to decide palindrome
*/
void palindrome()
{
  int d;
  char s[10];

  printf("Enter number for palindrome check: ");
  scanf("%s", s);

  int n = 0;
  for (int i=0; s[i]; i++)
    {
      d = s[i] - '0';
      n = n*10 + d;
      push(d);
    }

  int r = 0;
  while (top != -1)
    {
      d = pop();
      r = r*10 + d;
    }

  if ( r == n)
    printf("Palindrome number\n");
  else
    printf("Not Palindrome number\n");
}
```

```c
/*
1. Read and push the item
*/
void insert()
{
  int item;

  printf("Enter item to be pushed: ");
  scanf("%d", &item);

  push(item);
}

/*
1. Pop and print the item
*/
void delete()
{
  int item;

  item = pop();
  printf("Popped element is %d\n", item);
}

int main( )
{
  int choice;

  struct MENU
  {
    char *name; void (*func)();
  } menu[] = {
    {"Done", exit},
    {"Push", insert},
    {"Pop", delete},
    {"Display", display},
    {"Palindrome", palindrome}
  };

  int size = sizeof(menu)/sizeof(menu[0]);

  for(;;)
    {
      printf("\n-------Stack Menu-------\n");
      for(int i=1; i < size; i++)
        {
          printf("%d. %s\n", i, menu[i].name);
        }
      printf("---------------------------------\n");
```

```
      printf("Enter your choice: ");
      scanf("%d", &choice);

      menu[choice].func();
    }

  return 0;
}
```

4 Postfix

Design, Develop and Implement a Program in C for converting an Infix Expression to Postfix Expression. Program should support for both parenthesized and free parenthesized expressions with the operators: +, -, *, /, % (Remainder), ^ (Power) and alphanumeric operands.

4.1 Test cases

1. a*(b+c/d)-1 to abcd/+*1-
2. a+b+c-d to ab+c+d-
3. a^3*b to a3^b*
4. (a+b)*d+e/(f+a+d)+c to ab+d*efa+d+/+c+
5. ((a/(b-c+d))*(c-a)*c) to abc-d+/ca-**c**

4.2 Execution steps

1. gcc 4.c
2. ./a.out

4.3 Program

```c
/* 4.c Polish */
#include <stdio.h>
#define SIZE 50

/*
1. Create stack of characters
2. Implement push() and pop()
*/
char s[SIZE];
int top= -1;

void push(char elem)
{
  s[++top]=elem;
}

char pop()
{
  return(s[top--]);
}

void display(int k, char p[])
{
  for (int i=0; i <= top; i++)
    printf("%c", s[i]);
  printf("\n");

  for (int i=0; i < k; i++)
    printf("%c", p[i]);
}

/*
1. Define precedences for the operators
2. The operators are +, -, *, /, ^
3. And lowest for '('
4. Return lowest as default.
*/
int precedence(char elem)
{
  switch(elem)
    {
    case '(': return 1;
    case '+':
    case '-': return 2;
    case '*':
    case '/':
```

```
    case '%': return 3;
    case '^': return 4;

    default: return 0;
    }
}

/*
1. Input is infix expression
2. Output is postfix e* 6
3. Push the marker '\0'
4. Handle parens, operators and operands
*/
void convert(char infix[], char postfix[])
{
  char ch;
  int k=0;

  push('\0'); // null marker

  for (int i=0; infix[i]; i++)
    {
      ch = infix[i];
      switch(ch)
        {
        case '(':
          push(ch);
          break;

        case ')':
          while((ch = pop()) != '(')
            {
              postfix[k++] = ch;
            }
          break;

        case '+':
        case '-':
        case '/':
        case '*':
        case '%':
        case '^':
          while(precedence(ch) <= precedence(s[top]))
            {
              postfix[k++]=pop();
            }
          push(ch);
          break;
```

```
        default:
          postfix[k++]=ch;
        }
      //display(k, postfix);
    }

  // empty stack into postfix
  // Check if '\0' is at bottom of the stack

  while((postfix[k++] = pop()) != '\0');
}

/*
1. Declare infix and postfix arrays
2. Read infix expression
3. Convert infix to postfix expression
*/
int main()
{
  char infix[100];
  char postfix[100];

  printf("Enter the Infix Expression: ");
  gets(infix);

  convert(infix, postfix);
  printf("Postfix Expn: %s\n", postfix);
}
```

5 Expr

Design, Develop and Implement a Program in C for the following Stack Applications

```
a. Evaluation of Suffix expression with single digit operands and operators:
   +, -, *, /, %, ^
b. Solving Tower of Hanoi problem with n disks
```

5.1 Test cases

1. 23+ is 5
2. 23- is -1
3. `23^1-` is 7

Number of disks = 2

```
b --> a
b --> e
a --> e
```

5.2 Execution steps

1. gcc 5.c -lm
2. ./a.out

5.3 Program

```c
/* 5.c Expr */
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <math.h>
#include <string.h>

#define SIZE 50 /* Size of Stack */

int stack[SIZE];
int top = -1;

void push(int elem)
{
  stack[++top]=elem;
}

int pop()
{
  return(stack[top--]);
}

/*
1. Input is postfix expression as string
2. Push operands until the operator
3. evaluate exp with this operator and operands on the stack
4. Push the result on to the stack
*/
void suffix()
{
  char exp[50];
  char ch;

  printf("Enter suffix exression:");
  scanf("%s", exp);

  for (int i=0; exp[i]; i++)
    {
      ch = exp[i];

      switch(ch)
        {
        case '+':
          push(pop()+pop());
          break;
```

```
        case '-':
          push(pop()-pop());
          break;

        case '*':
          push(pop()*pop());
          break;

        case '/':
          push(pop()/pop());
          break;

        case '%':
          push(pop()%pop());
          break;

        case '^': push(pow(pop(), pop()));
          break;

        default:
          push(ch-'0'); /* Push the operand */
        }
    }

  printf("Result: %d, top=%d\n\n", pop(), top);
}

/*
1. Inputs are number of disks and 3 pegs 'b', 'a', and 'e'
2. Base case when n = 1, move b -> e
3. when n=2, b -> e, b -> a, a -> e
*/

void toh(int n, char beg, char aux, char end)
{
  if(n == 1)
    {
      printf("%c --> %c\n", beg, end);
      return;
    }
  toh(n-1, beg, end, aux);
  toh(1, beg, aux, end);
  toh(n-1, aux, beg, end);
}

void tower()
{
  int n;
```

```
  printf("Enter number of disks: ");
  scanf("%d", &n);

  toh(n, 'b', 'a', 'e');
}

/*
1. Evaluate postfix expression one or more times
2. Stop evaluation when the expressio is '.'
3. Solve tower of hanoi until the number diskes is 0
*/
int main()
{
  int choice;

  struct MENU
  {
    char *name; void (*func)();
  } menu[] = {
    {"Done",   exit  },
    {"Suffix", suffix},
    {"Tower",  tower }
  };

  int size = sizeof(menu)/sizeof(menu[0]);

  for(;;)
    {
      printf("\n-------Array Menu-------\n");
      for(int i=1; i < size; i++)
        {
          printf("%d. %s\n", i, menu[i].name);
        }
      printf("--------------------------------\n");

      printf("Enter your choice: ");
      scanf("%d", &choice);

      menu[choice].func();
    }

  return 0;
}
```

6 CQueue

Design, Develop and Implement a menu driven Program in C for the following operations on Circular QUEUE of Characters (Array Implementation of Queue with maximum size MAX)

```
a. Insert an Element on to Circular QUEUE
b. Delete an Element from Circular QUEUE
c. Demonstrate Overflow and Underflow situations on Circular QUEUE
d. Display the status of Circular QUEUE
e. Exit
```

Support the program with appropriate functions for each of the above operations

6.1 Test cases

1. Insert 1 2 3 4 one at a time and verify using display
2. Insert 5, Queue Full
3. Delete
4. Display 2 3 4
5. Insert 5
6. Display 2 3 4 5
7. Delete 2 3 4 5 one at a time
8. Delete any further should show empty

6.2 Execution steps

1. gcc -g 6.c
2. ./a.out

6.3 Program

```c
/* 6.c CQueue */
#include<stdio.h>
#include<stdlib.h>

#define MAXSIZE 5

int cq[MAXSIZE] = {0, 0, 0, 0, 0};
int front = 0, rear = 0;

/*
1. Read the item
2. Check if the queue is full i.e r+1 = f
3. increment as r = r+1 % MAXSIZE
4. Add the item
*/
void insert()
{
  int item;

  printf("Enter item: ");
  scanf("%d", &item);

  if ((rear + 1) %MAXSIZE == front)
    {
      printf("Circular queue is full.\n");
      return;
    }

  rear = (rear+1) % MAXSIZE;
  cq[rear] = item;
}

/*
1. Check if queue is empty i.e f = r
2. Increment front as f = f+1 % MAXSIZE
*/
void delete()
{
  int item;

  if (rear == front)
    {
      printf("Circular queue is empty.\n");
      return;
    }
```

```
  front = (front + 1) % MAXSIZE;
  item = cq[front];
}

/*
1. Print the items from f+1 to r
*/
void display()
{
  int n;

  printf("front = %d, rear = %d, Queue = ", front, rear);
  int i=front;
  for(;;)
    {
      if (i == rear) break;
      i = (i+1) % MAXSIZE;
      printf("%d ", cq[i]);
    }
  printf("\n");
}

int main( )
{
  int choice;

  struct MENU
  {
    char *name; void (*func)();
  } menu[] = {
    {"Done", exit},
    {"Insert", insert},
    {"Delete", delete},
    {"Display", display}
  };

  int size = sizeof(menu)/sizeof(menu[0]);

  for(;;)
    {
      printf("\n-------CQ Menu-------\n");
      for(int i=1; i < size; i++)
        {
          printf("%d. %s\n", i, menu[i].name);
        }
      printf("---------------------------------\n");

      printf("Enter your choice: ");
      scanf("%d", &choice);
```

```
      menu[choice].func();
    }
}
```

7 SLL

Design, Develop and Implement a menu driven Program in C for the following operations on Singly Linked List (SLL) of Student Data with the fields: USN, Name, Branch, Sem, PhNo

```
a. Create a SLL of N Students Data by using front insertion.
b. Display the status of SLL and count the number of nodes in it
c. Perform Insertion / Deletion at End of SLL
d. Perform Insertion / Deletion at Front of SLL(Demonstration of stack)
e. Exit
```

7.1 Test cases

1. Display empty list
2. Insert U1, U2, U3
3. Display list U3, U2, U1
4. Insert U4
5. Display U3, U2, U1, U4
6. Delete U4
7. Display U3, U2, U1
8. Insert U4
9. Display U4, U3, U2, U1

7.2 Execution steps

1. gcc -g 7.c
2. ./a.out

7.3 Program

```c
/* 7.c SLL */
#include <stdio.h>
#include <stdlib.h>

typedef struct NODE Node;

struct NODE
{
  char usn[20];
  char name[20];
  char branch[20];
  char sem[20];
  char phone[20];
  Node *next;
};

Node *start = NULL;
int flag=0; /* is 1 when list is created */

void get(char *prompt, char *s)
{
  printf("Enter %s:", prompt);
  scanf("%s", s);
}

Node *info()
{
  Node *s;

  s = malloc(sizeof(Node));

  get("USN", s->usn);
  get("Name", s->name);
  get("Branch", s->branch);
  get("Semester", s->sem);
  get("Phone", s->phone);

  s->next = NULL;
  return(s);
}

void insert_front()
{
   Node *student;

  student = info();
```

```
  if (start == NULL)
    {
      student->next = NULL;
      start = student;
      return;
    }

  student->next = start;
  start = student;
}

void insert_end()
{
  Node *p, *student;

  student = info();
  student->next = NULL;

  if (start == NULL)
    {
      start = student;
      return;
    }

  p = start;
  while(p->next != NULL)
    {
      p = p->next;
    }
  p->next = student;
}

void delete_front()
{
  Node *p;

  if (start == NULL)
    {
      printf("The list is empty. \n");
      return;
    }

  p = start;
  start = start->next;
  printf("Deleted Student USN is %s \n\n", p->usn);

  free(p);
}
```

```c
void delete_end()
{
  Node *p, *q;

  if (start == NULL)
    {
      printf("The list is empty. \n");
      return;
    }

  if (start->next == NULL)
    {
      printf("Deleted Student USN is %s \n\n", start->usn);
      free(start);
      start=NULL;
      return;
    }

  p = start;
  while (p->next != NULL)
    {
      q = p;
      p = p->next;
    }

  q->next = p->next;

  printf("Deleted Student USN is %s \n\n", p->usn);
  free(p);
}

void display()
{
  Node *p;
  int n;

  p = start;

  n = 1;
  while(p != NULL)
    {
      printf("%d. %s %s %s %s %s\n", n, p->usn, p->name, p->branch, p->sem, p->phone);
      p = p->next;
      n++;
    }
}

void create()
```

```
{
  int n;

  printf("Enter number of students: ");
  scanf("%d", &n);

  for(int i=1; i <= n; i++)
    {
      printf("\nStudent %d details\n", i);
      insert_front();
    }
}

int main()
{
  int choice;

  struct MENU
  {
    char *name;
    void (*func)();
  } menu[] = {
    {"Done", exit},
    {"Create", create},
    {"Insert front", insert_front},
    {"Insert end", insert_end},
    {"Delete front", delete_front},
    {"Delete end", delete_end},
    {"Display", display}
  };

  int size = sizeof(menu)/sizeof(menu[0]);

  for(;;)
    {
      printf("\n-------SLL Menu-------\n");
      for(int i=1; i < size; i++)
        {
          printf("%d. %s\n", i, menu[i].name);
        }
      printf("--------------------------------\n");

      printf("Enter your choice: ");
      scanf("%d", &choice);

      menu[choice].func();
    }
}
```

8 DLL

Design, Develop and Implement a menu driven Program in C for the following operations on Doubly Linked List (DLL) of Employee Data with the fields: SSN, Name, Dept, Designation, Sal, PhNo

```
a. Create a DLL of N Employees Data by using end insertion.
b. Display the status of DLL and count the number of nodes in it
c. Perform Insertion and Deletion at End of DLL
d. Perform Insertion and Deletion at Front of DLL
e. Demonstrate how this DLL can be used as Double Ended Queue.
f. Exit
```

Support the program with appropriate functions for each of the above operations

8.1 Test cases

1. Display empty list
2. Insert S1, S2, S3
3. Display list S3, S2, S1
4. Insert S4
5. Display S3, S2, S1, S4
6. Delete S4
7. Display S3, S2, S1
8. Insert S4
9. Display S4, S3, S2, S1

8.2 Execution steps

1. gcc 8.c
2. ./a.out

8.3 Program

```c
/* 8.c DLL */
#include <stdio.h>
#include <stdlib.h>

typedef struct NODE Node;
struct NODE
{
  char ssn[20];
  char name[20];
  char dept[20];
  char desig[20];
  char salary[10];
  char phone[12];
  Node *next;
  Node *prev;
};


void get(char *prompt, char *s)
{
  printf("Enter %s:", prompt);
  scanf("%s", s);
}

Node *start = NULL;
Node *info()
{
  Node *s;

  s = malloc(sizeof(Node));

  printf("Enter s info:");
  get("SSN", s->ssn);
  get("Name", s->name);
  get("Dept", s->dept);
  get("Designation", s->desig);
  get("Salary", s->salary);
  get("Phone", s->phone);

  s->next = NULL;

  return(s);
}

void insert_front()
{
```

```
  Node *student;

  student = info();

  if(start == NULL)
    {
      student->next = NULL;
      student->prev = NULL;
      start = student;
      return;
    }

  student->next = start;
  student->prev = NULL;

  start->prev = student;
  start = student;
}

void insert_end()
{
  Node *p, *student;

  student = info();

  if(start == NULL)
    {
      start = student;
      student->prev = NULL;
      return;
    }

  p = start;
  while(p->next != NULL)
    {
      p = p->next;
    }
  p->next = student;
  student->prev = p;
}

void delete_front()
{
  Node *p;

  if(start == NULL)
    {
      printf("The DLL list is empty. \n");
      return;
```

```
    }

  if (start->next == NULL)
    {
      printf("Deleted Employee SSN is %s \n\n",start->ssn);

      free(start);
      start = NULL;

      return;
    }

  p = start;
  start = start->next;
  start->prev = NULL;
  printf("Deleted Employee SSN is %s \n\n", p->ssn);

  free(p);
}

void delete_end()
{
  Node *p, *q;

  if(start == NULL)
    {
      printf("The DLL list is empty. \n");
      return;
    }

  if(start->next == NULL)
    {
      printf("Deleted Employee SSN is %s \n\n",start->ssn);
      free(start);
      start = NULL;
      return;
    }

  p = start;
  while(p->next != NULL)
    {
      p = p->next;
    }

  q = p->prev;
  q->next = NULL;

  printf("Deleted Employee SSN is %s \n\n", p->ssn);
  free(p);
```

```
}

void display()
{
  int n=0;
  Node *p;

  p = start;
  while(p != NULL)
    {
      n++;
      printf("%d. %s %s %s %s %s %s\n", n, p->ssn, p->name, p->dept, p->desig, p->salary, p->phon
      p = p->next;
    }
}

void create()
{
  int n;

  printf("Enter number of Employees: ");
  scanf("%d", &n);

  for(int i=1; i <= n; i++)
    {
      printf("\nEmployee %d details\n", i);
      insert_end();
    }
}

int main( )
{
  int choice;

  struct STRUCT
  {
    char *name; void (*func)();
  } menu[] = {
    {"Done", exit},
    {"Create", create},
    {"Insert front", insert_front},
    {"Insert end", insert_end},
    {"Delete front", delete_front},
    {"Delete end", delete_end},
    {"Display", display}
  };

  int size = sizeof(menu)/sizeof(menu[0]);
```

```
  for(;;)
    {
      printf("\n-------DLL Menu-------\n");
      for(int i=1; i < size; i++)
        {
          printf("%d. %s\n", i, menu[i].name);
        }
      printf("---------------------------------\n");

      printf("Enter your choice: ");
      scanf("%d", &choice);

      menu[choice].func();
    }
}
```

9 Poly

Design, Develop and Implement a Program in C for the following operations on Singly Circular Linked List (SCLL) with header nodes

```
a. Represent and Evaluate a Polynomial P(x,y,z) = 6x2y2z-4yz5+3x3yz+2xy5z-2xyz3
b. Find the sum of two polynomials POLY1(x,y,z) and POLY2(x,y,z) and store the result in POLYSUM(
```

9.1 Test cases

1. P(x,y,z) = 6x2y2z - 4yz5 + 3x3yz + 2xy5z - 2xyz3
   5
   6 2 2 1 -4 0 1 5 3 3 1 1 2 1 5 1 -2 1 1 3
   3 2 1
   With x = 3, y = 2, z = 1 p(x,y,z) = 550

1. P1(x,y,z) = 6x2y2z - 4yz5 + 3x3y, P2(x,y,z) = +2xy5z - 2xyz3, P1+P2?
   3 6 2 2 1 4 0 1 5 3 3 1 0
   2 2 1 5 1 -2 1 1 3

9.2 Execution steps

1. gcc -g 9.c
2. ./a.out

9.3 Program

```c
/* 9.c Poly */
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

typedef struct NODE Node;
struct NODE
{
  int coef;
  int xpow;
  int ypow;
  int zpow;
  Node *next;
};

/*
1. Traverse the header list
2. Start node is head->next
3. Stop when head is reached
*/
void display(Node *head)
{
  Node *p;

  p = head->next;
  while (p != head)
    {
      char sign = p->coef < 0 ? '-' : '+';

      printf("%c", sign);
      printf("%d", p->coef);
      if (p->xpow > 0) printf("x^%d", p->xpow);
      if (p->ypow > 0) printf("y^%d", p->ypow);
      if (p->zpow > 0) printf("z^%d", p->zpow);

      p = p->next;
    }
  printf("\n\n");
}

/*
1. Create the node and initialize its fields
2. Insert the node at the end of the header list
*/
void attach(Node *head, int coef, int xpow, int ypow, int zpow)
{
```

```
  Node *node = malloc(sizeof(Node));

  node->coef = coef;
  node->xpow = xpow;
  node->ypow = ypow;
  node->zpow = zpow;

  node->next = NULL;

  Node *p = head->next;
  while(p->next != head)
    {
      p = p->next;
    }
  node->next = head;
  p->next = node;
}

/*
1. Create header node
2. Get number of terms in the polynomial
3. Read data for each term
4. Attach the term to the given head
*/
Node *create()
{
  Node *head;
  int n;
  int coef, xpow, ypow, zpow;

  head = malloc(sizeof(Node));
  head->next = head;

  printf("Enter number of terms[0 if none]: ");
  scanf("%d", &n);

  for (int i=0; i<n; i++)
    {

      printf("Enter a term (coef, pows of x y z): ");
      scanf("%d%d%d%d", &coef, &xpow, &ypow, &zpow);

      attach(head, coef, xpow, ypow, zpow);
    }

  return(head);
}

/*
```

```
1. Create 3 circular lists with header nodes
2. Compare the terms of the two polynomials
3. Attach sum of the two polynomials as third polynomial:
   Case 1: sum of two polynomial terms if the exponents are same
   Case 2. 1st polynomial term if its x exponent is smaller
   Case 3. 2nd polynomial term if its x exponent is smaller
4. Advance two next term in the polynomials based on the above cases
*/
void padd()
{
  Node *h1 = create();
  Node *h2 = create();
  Node *h3 = create();

  printf("Create two polynomials.\n\n");

  display(h1);
  display(h2);
  display(h3);

  Node *p1 = h1->next, *p2 = h2->next;

  while(p1 != h1 && p2 != h2)
    {
      if( p1->xpow==p2->xpow && p1->ypow == p2->ypow && p1->zpow == p2->zpow)
        {
          attach(h3, p1->coef + p2->coef, p1->xpow, p1->ypow, p1->zpow);

          p1 = p1->next;
          p2 = p2->next;
        }
      else if(p1->xpow < p2->xpow)
        {
          attach(h3, p1->coef, p1->xpow, p1->ypow, p1->zpow);
          p1 = p1->next;
        }
      else
        {
          attach(h3, p2->coef, p2->xpow, p2->ypow, p2->zpow);
          p2 = p2->next;
        }

      display(h3);
    }

  while(p1 != h1)
    {
      attach(h3, p1->coef, p1->xpow, p1->ypow, p1->zpow);
      p1 = p1->next;
```

```
    }

  while(p2 != h2)
    {
      attach(h3, p2->coef, p2->xpow, p2->ypow, p2->zpow);
      p2 = p2->next;
    }

  display(h3);
}

/*
1. Create the polynomial of n terms
2. Read the values for x, y and z
3. Evaluate the valu of each terminal too find the sum
*/
void pval()
{
  int x, y, z;
  int value;

  Node *head = create();

  printf("Enter x y z: ");
  scanf("%d%d%d", &x, &y, &z);

  value = 0;

  Node *p = head->next;
  while(p != head)
    {
      value = value + (p->coef*pow(x, p->xpow)*pow(y, p->ypow)*pow(z, p->zpow));
      p = p->next;
    }

  printf("Polynomial value = %d \n", value);
}

int main()
{
  int choice;

  struct MENU
  {
    char *name; void (*func)();
  } menu[] = {
    {"Done", exit},
    {"Pval", pval},
    {"Padd", padd}
```

```
  };

  int size = sizeof(menu)/sizeof(menu[0]);

  for(;;)
    {
      printf("\n-------Menu-------\n");
      for(int i=1; i < size; i++)
        {
          printf("%d. %s\n", i, menu[i].name);
        }
      printf("---------------------------------\n");

      printf("Enter your choice: ");
      scanf("%d", &choice);

      menu[choice].func();
    }
}
```

10 BST

Design, Develop and Implement a menu driven Program in C for the following operations on Binary Search Tree (BST) of Integers.

```
a. Create a BST of N Integers: 6, 9, 5, 2, 8, 15, 24, 14, 7, 8, 5, 2
b. Traverse the BST in Inorder, Preorder and Post Order
c. Search the BST for a given element (KEY) and report the appropriate message
d. Exit
```

10.1 Test cases

```
     4
    /  \
   /    \
  2      5
 / \    / \
/   \  /   \
1   3  6   7

Create tree of 7 nodes 4 2 5 1 3 6 7
4 2 1 3 5 6 7 preorder
1 2 3 4 5 6 7 Inorder
1 3 2 6 7 5 4 postorder
find 7, found
find 8, not found
```

10.2 Execution steps

1. gcc -g 10.c
2. ./a.out

10.3 Program

```c
/* 10.c BST */
#include <stdio.h>
#include <stdlib.h>

typedef struct NODE Node;
struct NODE
{
  int value;
  Node *left;
  Node *right;
};

Node *root = NULL;

/*
1. If tree is empty, create tree with new node
2. If the value < root value then insert value as left child
3. If the value > root value then insert value as right child
*/

Node *insert(Node *root, int value)
{
  if (root == NULL)
    {
      Node *node = malloc(sizeof(Node));

      node->value = value;
      node->left = node->right = NULL;

      return node;
    }

  if (value == root->value)
    {
      printf(" Duplicate Element Not Allowed !!!");
    }

  if (value < root->value)
    {
      root->left = insert(root->left, value);
    }

  if (value > root->value)
    {
      root->right = insert(root->right, value);
    }
```

```
  return root;
}

/*
1. Return NULL if tree is empty
2. if value = root value, found the value
3. If value < root value search in left tree
4. If value > root value search in right tree
*/
Node *search(Node *root, int value)
{
  if(root == NULL) /* Element is not found */
    {
      return NULL;
    }

  if (value == root->value)
    {
      return root; /* Element Found */
    }

  if(value < root->value) /* Search in the left sub tree. */
    {
      return search(root->left, value);
    }

  if(value > root->value) /* Search in the right sub tree. */
    {
      return search(root->right, value);
    }
}

/*
1. Print left tree
2. Print root node
3. Print right tree
*/
void inorder(Node *root)
{
  if (root != NULL)
    {
      inorder(root->left);
      printf(" %d ", root->value);
      inorder(root->right);
    }
}

/*
```

```
1. Print root node
2. Print left tree
3. Print right tree
*/
void preorder(Node *root)
{
  if (root != NULL)
    {
      printf(" %d ", root->value);
      preorder(root->left);
      preorder(root->right);
    }
}

/*
1. Print left tree
2. Print right tree
3. Print root node
*/
void postorder(Node *root)
{
  if (root != NULL)
    {
      postorder(root->left);
      postorder(root->right);
      printf(" %d ", root->value);
    }
}

/*
1. Traverse the tree and print preorder, inorder and postorder sequences
*/
void traverse()
{
  preorder(root);
  printf(" Preorder\n");

  inorder(root);
  printf(" Inorder\n");

  postorder(root);
  printf(" Postorder\n");

}

/*
1. Read the value to search
2. Print the search result as Found or NOT found
*/
```

```
void find()
{
  Node *node;
  int value;

  printf("Enter item: ");
  scanf("%d", &value);

  node = search(root, value);

  node == NULL ? printf("Not found.\n") : printf("Found %d.\n", node->value);
}

/*
1. Read tree size and the node values
2. Create the tree by insertion
*/
void create()
{
  int n;
  int value;

  printf("BST for How Many Nodes? : ");
  scanf("%d", &n);

  for (int i=0; i<n; i++)
    {
      printf("Enter the value of node %d: ", i+1);
      scanf("%d", &value);
      root = insert(root, value);
    }

  printf("BST created!!\n");
}

int main()
{
  int choice;

  struct STRUCT
  {
    char *name; void (*func)();
  } menu[] = {
    {"Exit", exit},
    {"Create", create},
    {"traverse", traverse},
    {"Search", find}
  };
```

```
  int size = sizeof(menu)/sizeof(menu[0]);

  for(;;)
    {
      printf("\n-------BST Menu-------\n");
      for(int i=1; i < size; i++)
        {
          printf("%d. %s\n", i, menu[i].name);
        }
      printf("--------------------------------\n");

      printf("Enter your choice: ");
      scanf("%d", &choice);

      (*menu[choice].func)();
    }
}
```

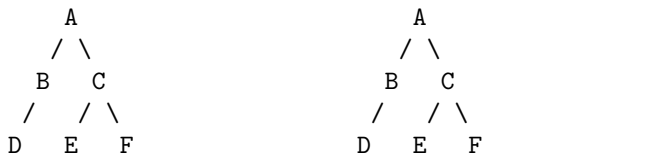
11 Graph

Design, Develop and Implement a Program in C for the following operations on Graph(G) of Cities

a. Create a Graph of N cities using Adjacency Matrix.
b. Print all the nodes reachable from a given starting node in a digraph using DFS/BFS method

11.1 Test cases

1) Test



BFS: A, B, C, D, E, F DFS: A, B, D, C, E, F

```
n=6
A B C D E F
0 1 1 0 0 0
0 0 0 1 0 0
0 0 0 0 1 1
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0

n = 6

0 1 1 0 0 0
0 0 0 1 0 0
0 0 0 0 1 1
0 1 0 0 0 0
0 0 1 0 0 0
0 0 1 0 0 0

DFS 0 1 3 2 4 5

Connected
```

2) Test 2

```
Repeat when F is not connected i.e (C,F) = 0

n = 6

0 1 1 0 0 0
0 0 0 1 0 0
0 0 0 0 1 0
0 1 0 0 0 0
0 0 1 0 0 0
0 0 1 0 0 0


DFS 0 1 3 2 4 5

NOT connected

3) Test 3

0-------3
| \
|  2
| / \
|/   \s
1     4

0 1 2 3 4

0 1 1 1 0
1 0 1 0 0
1 1 0 0 4
1 0 0 0 0
0 0 2 0 0
```

11.2 Execution steps

1. gcc -g 11.c
2. ./a.out

11.3 Program

```c
/* 11.c Graph */
#include<stdio.h>

int a[10][10];
int visited[10];
int n;

/*
1. explore the root vertex i
2. from i explore vertices (j) and process each vertex recursively
*/
void dfs(int i)
{
  visited[i] = 1;
  printf("%d ", i);

  for(int j=0; j < n; j++)
    {
      if(a[i][j] && !visited[j]) dfs(j);
    }
}

/*
1. count vertices visited and same as number of cities then G is connected
*/
void connected()
{
  int count = 0;

  for (int i=0; i < n; i++)
    {
      if(visited[i]) count++;
    }

  count == n ? printf("Connected.") : printf("NOT Connected.");
}

/*
1. Read number of cities
2. Read the graph as adjacency matrix
*/
void init()
{
  printf("Enter number of cities: ");
  scanf("%d", &n);
```

```
  printf("Enter graph a[][]:\n");
  for(int i=0; i < n; i++)
    {
      visited[i] = 0;
      for(int j=0; j < n; j++)
        {
          scanf("%d", &a[i][j]);
        }
    }
}

int main()
{
  init();

  printf("DFS ");
  dfs(0);

  connected();
}
```

12 bfc

Design, Develop and Implement a Program in C for the following operations on Graph(G) of Cities

a. Create a Graph of N cities using Adjacency Matrix.
b. Print all the nodes reachable from a given starting node in a digraph using DFS/BFS method

CODE:

```c
/* 11.c Graph */
#include<stdio.h>

int a[10][10];
int visited[10];
int n;

int q[10];
int f=-1, r=-1;

void addq(int n)
{
  q[++f] = n;
}

int delq()
{
  return q[++r];
}

int emptyq()
{
  return f==r;
}

void bfs(int i)
{
  addq(i);
  visited[i] = 1;

  while (!emptyq())
    {
      i = delq();
      printf("%d ", i);
```

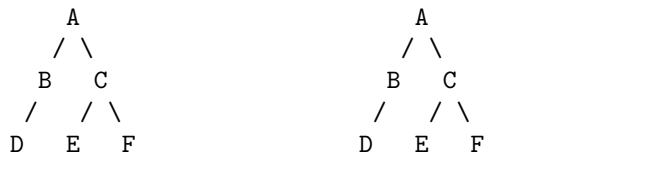
```
      for (int j=0; j < n; j++) {
        if (a[i][j] && !visited[j]) {

          addq(j);
          visited[j]=1;
        }
      }
    }
  printf("\n");
}

void connected()
{
  int count = 0;

  for (int i=0; i < n; i++)
    {
      if(visited[i]) count++;
    }

  count == n ? printf("Connected.") : printf("NOT Connected.");
}

void init()
{
  printf("Enter number of cities: ");
  scanf("%d", &n);

  printf("Enter graph a[][]:\n");
  for(int i=0; i < n; i++)
    {
      visited[i] = 0;
      for(int j=0; j < n; j++)
        {
          scanf("%d", &a[i][j]);
        }
    }
}

int main()
{
  init();

  printf("BFS ");
  bfs(0);

  connected();
}
```

Execution steps

1. gcc -g 11.c
2. ./a.out

Test cases

```
1) Test
```




```
BFS: A, B, C, D, E, F    DFS: A, B, D, C, E, F

n=6

A B C D E F
0 1 1 0 0 0
0 0 0 1 0 0
0 0 0 0 1 1
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0

DFS 0 1 2 3 4 5
```

13 File

Given a File of N employee records with a set K of Keys (4-digit) which uniquely determine the records in file F. Assume that file F is maintained in memory by a Hash Table (HT) of m memory locations with L as the set of memory addresses (2-digit) of locations in HT. Let the keys in K and addresses in L are Integers. Design and develop a Program in C that uses Hash function H: K őL as H(K)=K mod m (remainder method), and implement hashing technique to map a given key K to the address space L. Resolve the collision (if any) using linear probing.

13.1 Test cases

```
Enter number of records: 5
```

```
store n=0, key=1000, hk=6
store n=1, key=1005, hk=4
store n=2, key=1005, hk=4
 Collision.
store n=3, key=1010, hk=2
store n=4, key=1015, hk=0
Hash Table:
0       1015
1       -1
2       1010
3       -1
4       1005
5       1005
6       1000
```

13.2 Execution steps

1. gcc -g 12.c
2. ./a.out

Enter number of records: 5

13.3 Program

```c
/* 12.c File */
#include <stdio.h>
#include <stdlib.h>

#define m 7

int ht[m];

/*
1. Store key in HT at the index hk
2. if HT at hk is not avialable store next free location after hk
3. | 0 | 1 | 2 | 3 | 4 |, h = 2, h+5 % 5 = 2 wraps around after m
*/
void store(int key, int n)
{
  int pos = -1;
  int flag;

  if (ht[n] == -1)
    {
      ht[n] = key;
      return;
    }
  printf(" Collision.\n");

  // Linear probing
  for (int i = 0; i < m; i++)
    {
      n = ++n % m;
      if (ht[n] == -1)
        {
          ht[n] = key;
          return;
        }
    }
  printf(" Overflow?\n");
}

/*
1. Display the HT
*/
void display()
{
  printf("Hash Table:\n");
  for(int i = 0; i < m; i++)
    {
```

```
      printf("%d\t%d\n", i, ht[i]);
    }
}

void init()
{
  for(int i = 0; i < m; i++)
    {
      ht[i] = -1;
    }
}

/*
1. Initialize HT with empty values (say -1)
2. Generate 4 digits keys
3. Map 4 digit key to 2 digit hash key
4. Store the key at HT[hk]
*/
int main()
{
  FILE *fp;
  int key;
  int hk;
  int n;

  init();

  printf("Enter number of records: ", &n);
  scanf("%d", &n);

  fp = fopen("12.txt", "r");

  for (int i=0; i < n; i++)
    {
      fscanf(fp, "%d", &key);
      hk = key % m;
      printf("store n=%d, key=%d, hk=%d\n", i, key, hk);
      store(key, hk);
    }

  display();
}
```

FILE: 12.txt

```
1000
1005
```

```
1005
1010
1015
1020
```