****

**School of Computer Science and Engineering**

**Practical File**

**DATA STRUCTURE AND ALGORITHM**

**BCSE2361**

**SECTION : 03 (P1)**

**SCHOOL : SCSE**

**PROGRAM : B. TECH**

**SUBMITTED BY : ABHAY PRATAP SINGH PARIHAR (20SCSE1010167)**

**SUBMITTED TO :**

***INDEX***

|  |  |  |  |
| --- | --- | --- | --- |
| **Sno.** | **Experiment** | **Date** | **Remark** |
| 1 | Write and test a function that can help in Linear and Binary search. |  |  |
| 2 | Write and test a recursive function that prints all the permutations of the first n characters of a string. |  |  |
| 3 | Write and test a recursive function that returns the power xn |  |  |
| 4 | Write a program to implement a stack of strings (illustrate the operations push (), pop(), size(), empty() and top()). |  |  |
| 5 | Write a program to implement a Queue of strings (illustrate the operations Enqueue (), dequeue(), size(), full() and empty()). |  |  |
| 6 | Write a program to implement Bubble Sort. |  |  |
| 7 | Write a program to implement selection sort. |  |  |
| 8 | Write a program to implement Quick Sort. |  |  |

***Experiment-1***

**AIM :** Write and test a function that can help in Linear and Binary search.

1. Binary Search-

**Program-**

#include <stdio.h>

int main()

{

int c, first, last, middle, n, search, array[100];

printf("Enter number of elements\n");

scanf("%d", &n);

printf("Enter %d integers\n", n);

for (c = 0; c < n; c++)

scanf("%d", &array[c]);

printf("Enter value to find\n");

scanf("%d", &search);

first = 0;

last = n - 1;

middle = (first+last)/2;

while (first <= last) {

if (array[middle] < search)

first = middle + 1;

else if (array[middle] == search) {

printf("%d found at location %d.\n", search, middle+1);

break;

}

else

last = middle - 1;

middle = (first + last)/2;

}

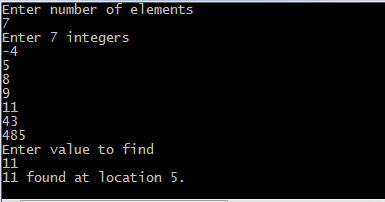
if (first > last)

printf("Not found! %d isn't present in the list.\n", search);

return 0;

}

**Output-**



1. Linear search-

**Program-**

#include <stdio.h>

int main()

{

int array[100], search, c, n;

printf("Enter number of elements in array\n");

scanf("%d", &n);

printf("Enter %d integer(s)\n", n);

for (c = 0; c < n; c++)

scanf("%d", &array[c]);

printf("Enter a number to search\n");

scanf("%d", &search);

for (c = 0; c < n; c++)

{

if (array[c] == search)

{

printf("%d is present at location %d.\n", search, c+1);

break;

}

}

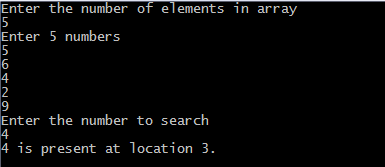
if (c == n)

printf("%d isn't present in the array.\n", search);

return 0;

}

**Output-**



***Experiment-2***

**AIM :**Write and test a recursive function that prints all the permutations of the first n characters of a string.

**Program-**

#include<stdio.h>

#include<string.h>

//Declaring generatePermutation()

void generatePermutation(char \* , int , int );

int main()

{

char str[] = "ABC";

int n =strlen(str);

printf("All the permutations of the string are: \n");

generatePermutation(str,0,n);

}

//Function for generating different permutation of the string.

void generatePermutation(char \*str,const int start, int end)

{

char temp;

int i,j;

for(i = start; i < end-1; ++i){

for(j = i+1; j < end; ++j)

{

//Swapping the string by fixing a character

temp = str[i];

str[i] = str[j];

str[j] = temp;

//Recursively calling function generatePermutation() for rest of the characters

generatePermutation(str , i+1 ,end);

//Backtracking and swapping the characters again

temp = str[i];

str[i] = str[j];

str[j] = temp;

}

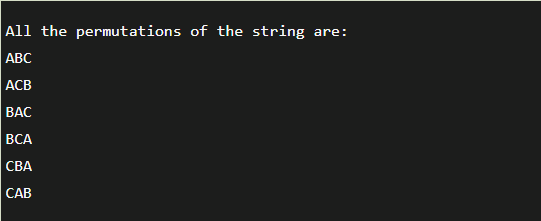
}

//Print the permutations

printf("%s\n",str);

}

**Output-**



***Experiment-3***

**AIM:** Write and test a recursive function that returns the power xn

**Program-**

#include <stdio.h>

int power(int n1, int n2);

int main() {

int base, a, result;

printf("Enter base number: ");

scanf("%d", &base);

printf("Enter power number(positive integer): ");

scanf("%d", &a);

result = power(base, a);

printf("%d^%d = %d", base, a, result);

return 0;

}

int power(int base, int a) {

if (a != 0)

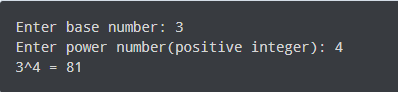
return (base \* power(base, a - 1));

else

return 1;

}

**Output-**



***Experiment-4***

**AIM:** Write a program to implement a stack of strings (illustrate the operations push (), pop(), size(), empty() and top()).

#include <stdio.h>

#include <stdlib.h>

// Data structure to represent a stack

struct stack

{

    int maxsize;    // define max capacity of the stack

    int top;

    int \*items;

};

// Utility function to initialize the stack

struct stack\* newStack(int capacity)

{

    struct stack \*pt = (struct stack\*)malloc(sizeof(struct stack));

    pt->maxsize = capacity;

    pt->top = -1;

    pt->items = (int\*)malloc(sizeof(int) \* capacity);

    return pt;

}

// Utility function to return the size of the stack

int size(struct stack \*pt) {

    return pt->top + 1;

}

// Utility function to check if the stack is empty or not

int isEmpty(struct stack \*pt) {

    return pt->top == -1;                   // or return size(pt) == 0;

}

// Utility function to check if the stack is full or not

int isFull(struct stack \*pt) {

    return pt->top == pt->maxsize - 1;      // or return size(pt) == pt->maxsize;

}

// Utility function to add an element `x` to the stack

void push(struct stack \*pt, int x)

{

    // check if the stack is already full. Then inserting an element would

    // lead to stack overflow

    if (isFull(pt))

    {

        printf("Overflow\nProgram Terminated\n");

        exit(EXIT\_FAILURE);

    }

    printf("Inserting %d\n", x);

    // add an element and increment the top's index

    pt->items[++pt->top] = x;

}

// Utility function to return the top element of the stack

int peek(struct stack \*pt)

{

    // check for an empty stack

    if (!isEmpty(pt)) {

        return pt->items[pt->top];

    }

    else {

        exit(EXIT\_FAILURE);

    }

}

// Utility function to pop a top element from the stack

int pop(struct stack \*pt)

{

    // check for stack underflow

    if (isEmpty(pt))

    {

        printf("Underflow\nProgram Terminated\n");

        exit(EXIT\_FAILURE);

    }

    printf("Removing %d\n", peek(pt));

    // decrement stack size by 1 and (optionally) return the popped element

    return pt->items[pt->top--];

}

int main()

{

    // create a stack of capacity 5

    struct stack \*pt = newStack(5);

    push(pt, 1);

    push(pt, 2);

    push(pt, 3);

    printf("The top element is %d\n", peek(pt));

    printf("The stack size is %d\n", size(pt));

    pop(pt);

    pop(pt);

    pop(pt);

    if (isEmpty(pt)) {

        printf("The stack is empty");

    }

    else {

        printf("The stack is not empty");

    }

    return 0;

    }

**Output-**

Inserting 1  
Inserting 2  
Inserting 3  
The top element is 3  
The stack size is 3  
Removing 3  
Removing 2  
Removing 1  
The stack is empty

***Experiment-5***

**AIM:** Write a program to implement a Queue of strings (illustrate the operations Enqueue (), dequeue(), size(), full() and empty()).

**Program-**

#include <limits.h>

#include <stdio.h>

#include <stdlib.h>

// A structure to represent a queue

struct Queue {

    int front, rear, size;

    unsigned capacity;

    int\* array;

};

// function to create a queue

// of given capacity.

// It initializes size of queue as 0

struct Queue\* createQueue(unsigned capacity)

{

    struct Queue\* queue = (struct Queue\*)malloc(

        sizeof(struct Queue));

    queue->capacity = capacity;

    queue->front = queue->size = 0;

    // This is important, see the enqueue

    queue->rear = capacity - 1;

    queue->array = (int\*)malloc(

        queue->capacity \* sizeof(int));

    return queue;

}

// Queue is full when size becomes

// equal to the capacity

int isFull(struct Queue\* queue)

{

    return (queue->size == queue->capacity);

}

// Queue is empty when size is 0

int isEmpty(struct Queue\* queue)

{

    return (queue->size == 0);

}

// Function to add an item to the queue.

// It changes rear and size

void enqueue(struct Queue\* queue, int item)

{

    if (isFull(queue))

        return;

    queue->rear = (queue->rear + 1)

                  % queue->capacity;

    queue->array[queue->rear] = item;

    queue->size = queue->size + 1;

    printf("%d enqueued to queue\n", item);

}

// Function to remove an item from queue.

// It changes front and size

int dequeue(struct Queue\* queue)

{

    if (isEmpty(queue))

        return INT\_MIN;

    int item = queue->array[queue->front];

    queue->front = (queue->front + 1)

                   % queue->capacity;

    queue->size = queue->size - 1;

    return item;

}

// Function to get front of queue

int front(struct Queue\* queue)

{

    if (isEmpty(queue))

        return INT\_MIN;

    return queue->array[queue->front];

}

// Function to get rear of queue

int rear(struct Queue\* queue)

{

    if (isEmpty(queue))

        return INT\_MIN;

    return queue->array[queue->rear];

}

// Driver program to test above functions./

int main()

{

    struct Queue\* queue = createQueue(1000);

    enqueue(queue, 10);

    enqueue(queue, 20);

    enqueue(queue, 30);

    enqueue(queue, 40);

    printf("%d dequeued from queue\n\n",

           dequeue(queue));

    printf("Front item is %d\n", front(queue));

    printf("Rear item is %d\n", rear(queue));

    return 0;

}

**Output-**

10 enqueued to queue

20 enqueued to queue

30 enqueued to queue

40 enqueued to queue

10 dequeued from queue

Front item is 20

Rear item is 40

***Experiment-6***

**AIM:** Write a program to implement Bubble Sort.

**Program-**

#include<stdio.h>

void print(int a[], int n) //function to print array elements

{

int i;

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

{

printf("%d ",a[i]);

}

}

void bubble(int a[], int n) // function to implement bubble sort

{

int i, j, temp;

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

{

for(j = i+1; j < n; j++)

{

if(a[j] < a[i])

{

temp = a[i];

a[i] = a[j];

a[j] = temp;

}

}

}

}

void main ()

{

int i, j,temp;

int a[5] = { 10, 35, 32, 13, 26};

int n = sizeof(a)/sizeof(a[0]);

printf("Before sorting array elements are - \n");

print(a, n);

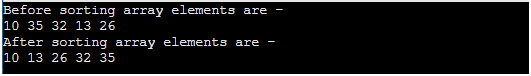
bubble(a, n);

printf("\nAfter sorting array elements are - \n");

print(a, n);

}

**Output-**



***Experiment-7***

**AIM:** Write a program to implement selection sort.

**Program-**

#include <stdio.h>

void selection(int arr[], int n)

{

int i, j, small;

for (i = 0; i < n-1; i++) // One by one move boundary of unsorted subarray

{

small = i; //minimum element in unsorted array

for (j = i+1; j < n; j++)

if (arr[j] < arr[small])

small = j;

// Swap the minimum element with the first element

int temp = arr[small];

arr[small] = arr[i];

arr[i] = temp;

}

}

void printArr(int a[], int n) /\* function to print the array \*/

{

int i;

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

printf("%d ", a[i]);

}

int main()

{

int a[] = { 12, 31, 25, 8, 32, 17 };

int n = sizeof(a) / sizeof(a[0]);

printf("Before sorting array elements are - \n");

printArr(a, n);

selection(a, n);

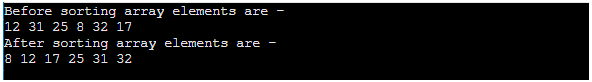
printf("\nAfter sorting array elements are - \n");

printArr(a, n);

return 0;

}

Output-



***Experiment-8***

**AIM:** Write a program to implement Quick Sort.

**Program-**

#include <stdio.h>

/\* function that consider last element as pivot,

place the pivot at its exact position, and place

smaller elements to left of pivot and greater

elements to right of pivot. \*/

int partition (int a[], int start, int end)

{

int pivot = a[end]; // pivot element

int i = (start - 1);

for (int j = start; j <= end - 1; j++)

{

// If current element is smaller than the pivot

if (a[j] < pivot)

{

i++; // increment index of smaller element

int t = a[i];

a[i] = a[j];

a[j] = t;

}

}

int t = a[i+1];

a[i+1] = a[end];

a[end] = t;

return (i + 1);

}

/\* function to implement quick sort \*/

void quick(int a[], int start, int end) /\* a[] = array to be sorted, start = Starting index, end = Ending index \*/

{

if (start < end)

{

int p = partition(a, start, end); //p is the partitioning index

quick(a, start, p - 1);

quick(a, p + 1, end);

}

}

/\* function to print an array \*/

void printArr(int a[], int n)

{

int i;

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

printf("%d ", a[i]);

}

int main()

{

int a[] = { 24, 9, 29, 14, 19, 27 };

int n = sizeof(a) / sizeof(a[0]);

printf("Before sorting array elements are - \n");

printArr(a, n);

quick(a, 0, n - 1);

printf("\nAfter sorting array elements are - \n");

printArr(a, n);

return 0;

}

**Output-**

