MODULE-5

R1. PERMUTATION OF STRINGS

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

int next\_permutation(int n, char \*\*s)

{

int i,j;

int k = -1;

for ( i = 0; i < n-1; i++) {

if (strcmp(s[i], s[i+1]) < 0)

k = i;

}

if (k == -1) return 0;

int l = -1;

for ( i = k+1; i < n; i++) {

if (strcmp(s[k], s[i]) < 0)

l = i;

}

char \*tmp = s[k];

s[k] = s[l];

s[l] = tmp;

i = k+1, j = n-1;

while (i < j) {

tmp = s[i];

s[i++] = s[j];

s[j--] = tmp;

}

return 1;

}

int main()

{

char \*\*s;

int n,i;

scanf("%d", &n);

s = calloc(n, sizeof(char\*));

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

{

s[i] = calloc(11, sizeof(char));

scanf("%s", s[i]);

}

do

{

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

printf("%s%c", s[i], i == n - 1 ? '\n' : ' ');

} while (next\_permutation(n, s));

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

free(s[i]);

free(s);

return 0;

}

R2. 2D ARRAYS

#include <assert.h>

#include <ctype.h>

#include <limits.h>

#include <math.h>

#include <stdbool.h>

#include <stddef.h>

#include <stdint.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

char\* readline();

char\* ltrim(char\*);

char\* rtrim(char\*);

char\*\* split\_string(char\*);

int parse\_int(char\*);

int main()

{

int i,j,k;

int arr[6][6],temp=-9999,a,b;

for(i=0;i<6;i++)

for(j=0;j<6;j++)

scanf("%d",&arr[i][j]);

for(i=0;i<=3;i++)

for(j=0;j<=3;j++)

{

}

a =arr[i][j]+arr[i][j+1]+arr[i][j+2]+arr[i+1][j+1]+arr[i+2][j]+arr[i+2][j+1]+arr[i+2][j+2];

if(temp < a)

temp = a ;

printf("%d",temp);

return 0;

}

char\* readline() {

size\_t alloc\_length = 1024;

size\_t data\_length = 0;

char\* data = malloc(alloc\_length);

while (true) {

char\* cursor = data + data\_length;

char\* line = fgets(cursor, alloc\_length-data\_length, stdin);

if (!line) {

break;

}

data\_length += strlen(cursor);

if (data\_length < alloc\_length-1 || data[data\_length-1] == '\n') {

break;

}

alloc\_length <<= 1;

data = realloc(data, alloc\_length);

if (!data) {

data = '\0';

break;

}

}

if (data[data\_length-1] == '\n') {

data[data\_length-1] = '\0';

data = realloc(data, data\_length);

if (!data) {

data = '\0';

}

} else {

data = realloc(data, data\_length + 1);

if (!data) {

data = '\0';

} else {

data[data\_length] = '\0';

}

}

return data;

}

char\* ltrim(char\* str) {

if (!str) {

return '\0';

}

if (!\*str) {

return str;

}

while (\*str != '\0' && isspace(\*str)) {

str++;

}

return str;

}

char\* rtrim(char\* str) {

if (!str) {

return '\0';

}

if (!\*str) {

return str;

}

char\* end = str + strlen(str)-1;

while (end >= str && isspace(\*end)) {

end--;

}

\*(end + 1) = '\0';

return str;

}

char\*\* split\_string(char\* str) {

char\*\* splits = NULL;

char\* token = strtok(str, " ");

int spaces = 0;

while (token) {

splits = realloc(splits, sizeof(char\*) \* ++spaces);

if (!splits) {

return splits;

}

splits[spaces-1] = token;

token = strtok(NULL, " ");

}

return splits;

}

int parse\_int(char\* str) {

char\* endptr;

int value = strtol(str, &endptr, 10);

if (endptr == str || \*endptr != '\0') {

exit(EXIT\_FAILURE);

}

return value;

}

R3. DYNAMIC ARRAYS

#include <stdio.h>

#include <stdlib.h>

int main() {

int n, q,i=0;

scanf("%d %d", &n, &q);

// Create an array of dynamic arrays for the shelves

int\*\* shelves = (int\*\*)malloc(n \* sizeof(int\*));

int\* sizes = (int\*)malloc(n \* sizeof(int)); // To keep track of the number of books in each shelf

int last\_ans = 0;

// Initialize sizes

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

sizes[i] = 0;

shelves[i] = NULL; // Initialize each shelf to NULL

}

// Process each query

for ( i = 0; i < q; i++) {

int query\_type, x, y;

scanf("%d %d %d", &query\_type, &x, &y);

// Calculate the index for the shelf

int idx = (x ^ last\_ans) % n;

if (query\_type == 1) {

// Add a book with y pages to shelf idx

shelves[idx] = (int\*)realloc(shelves[idx], (sizes[idx] + 1) \* sizeof(int));

shelves[idx][sizes[idx]] = y; // Add the number of pages

sizes[idx]++; // Increment the count of books on shelf idx

} else if (query\_type == 2) {

// Retrieve the number of pages in the y-th book on shelf idx

last\_ans = shelves[idx][y % sizes[idx]];

printf("%d\n", last\_ans);

} else if (query\_type == 3) {

// Print the total number of books on shelf idx

printf("%d\n", sizes[idx]);

}

}

// Free allocated memory

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

free(shelves[i]); // Free each shelf

}

free(shelves); // Free the shelves array

free(sizes); // Free the sizes array

return 0;

}

A4. PRINTING TOKENS

#include <stdio.h>

#include <string.h>

#include <math.h>

#include <stdlib.h>

int main() {

char \*s;

int i;

s = malloc(1024 \* sizeof(char));

scanf("%[^\n]", s);

s = realloc(s, strlen(s) + 1);

for(i=0;i<strlen(s);i++){

if(\*(s+i)==' ')

printf("\n");

else

printf("%c",\*(s+i));

}

free(s);

return 0;

}

A5. FIND THE INDEX OF A FIRST OCCURRENCE IN A STRING

#include <stdio.h>

#include <string.h>

int main() {

char haystack[100];

char needle[100];

scanf("%s", haystack);

scanf("%s", needle);

int result = strStr(haystack, needle);

printf("l%d\n", result);

return 0;

}

int strStr(char\* haystack, char\* needle) {

int hsize = strlen(haystack);

int nsize = strlen(needle);

int res =-1;

int i = 0, j= 0;

while (haystack[i]!='\0' && needle[j]!='\0' ) {

if (haystack[i] == needle[j]) {

i++; j++;

}

else {

i++; j = 0;

}

}

if (j == nsize)

res =(i- nsize);

else

res=-1;

return res;

}

**MODULE - 4**

**1. REGULAR - SORTING ARRAY OF STRINGS**

#include<string.h>

#include<stdio.h>

#include<stdlib.h>

int lexicographic\_sort(const char\* a, const char\* b){

return strcmp(a, b) > 0;

}

int lexicographic\_sort\_reverse(const char\* a, const char\* b){

return strcmp(a, b) <= 0;

}

int sort\_by\_number\_of\_distinct\_characters(const char\* a, const char\* b){

int c1 = 0, c2 = 0;

int hsh1[26] = {0}, hsh2[26] = {0};

int n1 = strlen(a);

int n2 = strlen(b);

int i;

for(i = 0; i < n1; i++){

hsh1[a[i] - 'a'] = 1;

}

for(i = 0; i < n2; i++){

hsh2[b[i] - 'a'] = 1;

}

for( i = 0; i < 26; i++){

if(hsh1[i])

c1++;

if(hsh2[i])

c2++;

}

if( c1 != c2)

return c1 > c2;

else

return strcmp(a, b) > 0;

}

int sort\_by\_length(const char\* a, const char\* b){

if(strlen(a) != strlen(b))

return strlen(a) > strlen(b);

else

return strcmp(a, b) > 0;

}

void string\_sort(char\*\* arr,const int len,int (\*cmp\_func)(const char\* a, const char\* b))

{ int i;

for( i = 1; i < len; i++){

int j = i;

char\* p = arr[i];

while(j > 0){

if((\*cmp\_func)(arr[j-1],p) > 0 )

arr[j] = arr[j-1];

else

break;

j--;

}

arr[j] = p;

}

}

int main()

{

int n,i;

scanf("%d", &n);

char\*\* arr;

arr = (char\*\*)malloc(n \* sizeof(char\*));

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

\*(arr + i) = malloc(1024 \* sizeof(char));

scanf("%s", \*(arr + i));

\*(arr + i) = realloc(\*(arr + i), strlen(\*(arr + i)) + 1);

}

string\_sort(arr, n, lexicographic\_sort);

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

printf("%s\n", arr[i]);

printf("\n");

string\_sort(arr, n, lexicographic\_sort\_reverse);

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

printf("%s\n", arr[i]);

printf("\n");

string\_sort(arr, n, sort\_by\_length);

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

printf("%s\n", arr[i]);

printf("\n");

string\_sort(arr, n, sort\_by\_number\_of\_distinct\_characters);

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

printf("%s\n", arr[i]);

printf("\n");

}

****

**2. REGULAR - 1D ARRAYS IN C**

#include <stdio.h>

#include <stdlib.h>

int main()

{

int n;

scanf("%d", &n);

// Create a dynamic array of size n

int\* arr = (int\*)malloc(n \* sizeof(int));

// Read the values from stdin and store them in the array

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

scanf("%d", &arr[i]);

}

// Calculate the sum of all elements in the array

int sum = 0;

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

sum += arr[i];

}

// Print the sum

printf("%d\n", sum);

// Free the memory where the array is stored

free(arr);

return 0;

}

A screenshot of a computer

Description automatically generated

**3. REGULAR - Array Reversal**

#include <stdio.h>

#include <stdlib.h>

int main()

{

int num, \*arr, i;

scanf("%d", &num);

arr = (int\*) malloc(num \* sizeof(int));

for(i = 0; i < num; i++) {

scanf("%d", arr + i);

}

for(i = num-1; i>-1; i--)

printf("%d ", \*(arr + i));

return 0;

}

A screenshot of a black screen

Description automatically generated

**4. ADDITIONAL-Binary Search Tree: Insertion**

#include <stdio.h>

#include <string.h>

#include <math.h>

#include <stdlib.h>

struct node {

int data;

struct node \*left;

struct node \*right;

};

void preOrder( struct node \*root) {

if( root == NULL )

return;

printf("%d ",root->data);

preOrder(root->left);

preOrder(root->right);

}

struct node\* insert(struct node\* root, int data) {

if (root == NULL) {

struct node\* newNode = (struct node\*)malloc(sizeof(struct node));

newNode->data = data;

newNode->left = NULL;

newNode->right = NULL;

return newNode;

}

if (data < root->data) {

root->left = insert(root->left, data);

} else {

root->right = insert(root->right, data);

}

return root;

}

int main() {

struct node\* root = NULL;

int t;

int data;

scanf("%d", &t);

while(t-- > 0) {

scanf("%d", &data);

root = insert(root, data);

}

preOrder(root);

return 0;

}

A black screen with white text

Description automatically generated

**5. ADDITIONAL - REMOVING DUPLICATES FROM ARRAY**

#include <stdio.h>

#include <conio.h>

int main ()

{

// declare local variables

int arr[20], i, j, k, size;

printf (" Define the number of elements in an array: ");

scanf (" %d", &size);

printf (" \n Enter %d elements of an array: \n ", size);

// use for loop to enter the elements one by one in an array

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

{

scanf (" %d", &arr[i]);

}

// use nested for loop to find the duplicate elements in array

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

{

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

{

// use if statement to check duplicate element

if ( arr[i] == arr[j])

{

// delete the current position of the duplicate element

for ( k = j; k < size - 1; k++)

{

arr[k] = arr [k + 1];

}

// decrease the size of array after removing duplicate element

size--;

// if the position of the elements is changes, don't increase the index j

j--;

}

}

}

/\* display an array after deletion or removing of the duplicate elements \*/

printf (" \n Array elements after deletion of the duplicate elements: ");

// for loop to print the array

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

{

printf (" %d \t", arr[i]);

}

return 0;

}

A screenshot of a computer

Description automatically generated