**Module 4**

**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");

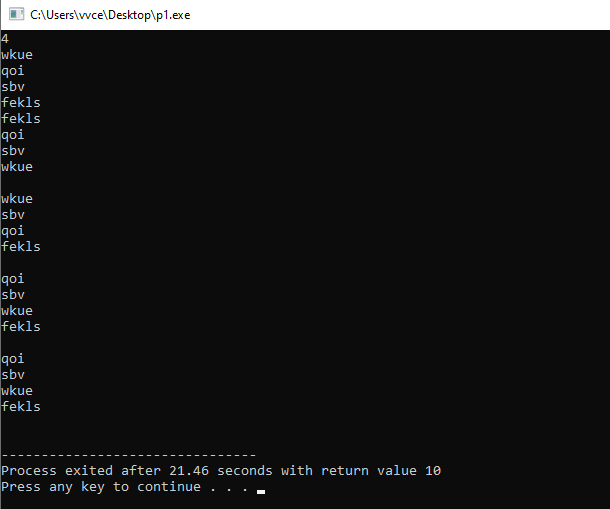
string\_sort(arr, n, sort\_by\_number\_of\_distinct\_characters);

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

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

printf("\n");

}

****

**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];

}

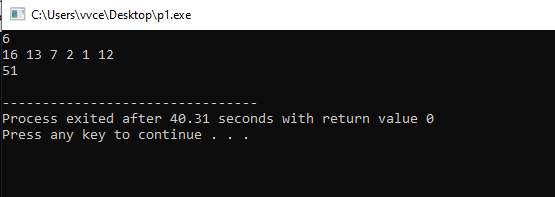
printf("%d\n", sum);

// Free the memory where the array is stored

free(arr);

return 0;

}



**Array Reversal**

#include <stdio.h>

#include <stdlib.h>

int main() {

int n, arr[1000], i;

scanf("%d", &n);

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

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

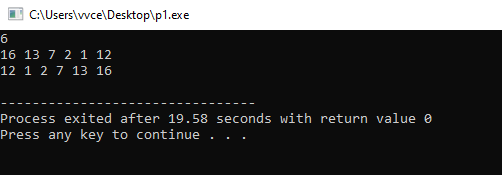
for (i = n - 1; i >= 0; i--)

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

printf("\n");

return 0;

}

****

**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);

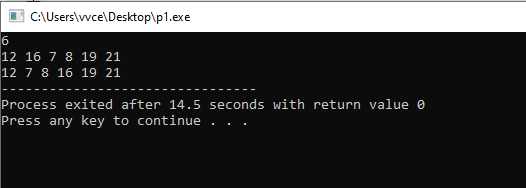
root = insert(root, data);

}

preOrder(root);

return 0;

}



**Remove Duplicates from Sorted Array**

#include <stdio.h>

// Function to remove duplicates from a sorted array

int removeDuplicates(int\* nums, int numsSize) {

// Edge case: if the array is empty, no unique elements exist

if (numsSize == 0) {

return 0;

}

// k will track the index of the last unique element

int k = 1;

int i;

// Start from the second element (index 1)

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

// If the current element is different from the previous one, it's unique

if (nums[i] != nums[i - 1]) {

// Place the unique element at position k

nums[k] = nums[i];

k++; // Increment k to track the number of unique elements

}

}

// Return the number of unique elements

return k;

}

// Main function to test the removeDuplicates function

int main() {

// Example input

int nums[] = {1, 1, 2, 2, 3, 3, 4};

int i;

int numsSize = sizeof(nums) / sizeof(nums[0]);

// Calling removeDuplicates function

int newSize = removeDuplicates(nums, numsSize);

// Print the modified array and the number of unique elements

printf("Array after removing duplicates: ");

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

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

}

printf("\n");

printf("Number of unique elements: %d\n", newSize);

return 0;

}

