Day -4

Assignment

## Menda Mani Sai

## 192111399

## 1. Write a program to inset a number in a list.

#include <stdio.h>

int main() {

int originalArray[100];

int newArray[101];

int n, num, pos, i;

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

scanf("%d", &n);

printf("Enter %d elements:\n", n);

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

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

}

printf("Enter the number to insert: ");

scanf("%d", &num);

printf("Enter the position to insert (0-%d): ", n);

scanf("%d", &pos);

if (pos < 0 || pos > n) {

printf("Invalid position. Position should be between 0 and %d\n", n);

return 1;

}

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

newArray[i] = originalArray[i];

}

newArray[pos] = num;

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

newArray[i + 1] = originalArray[i];

}

n++;

printf("Updated array:\n");

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

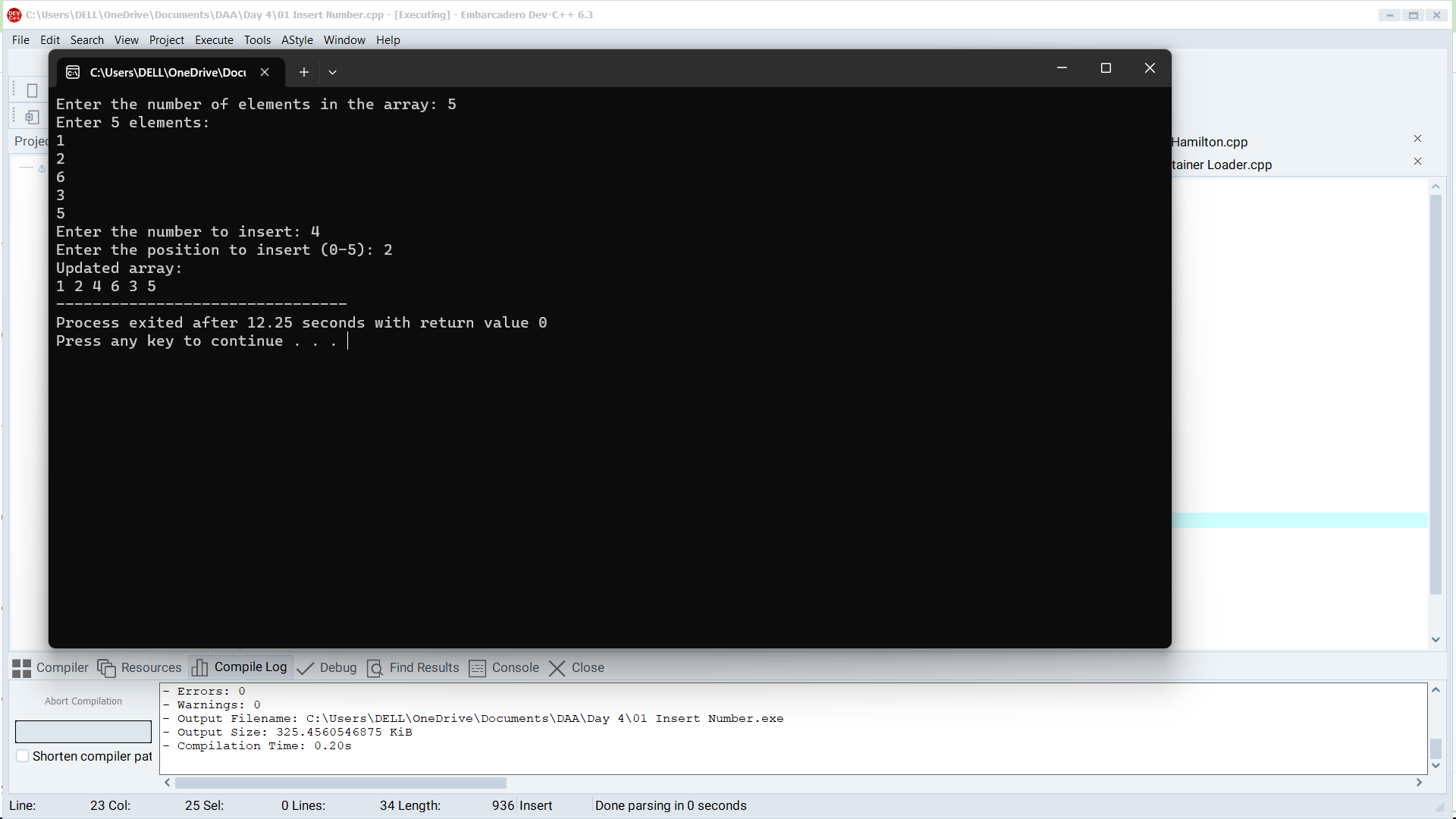
printf("%d ", newArray[i]);

}

return 0;

}

#### OUTPUT:



## 2. Write a program to perform sum of subsets problem using backtracking.

#include <stdio.h>

#define MAX\_SIZE 100

int set[MAX\_SIZE];

int solution[MAX\_SIZE];

int n, targetSum;

void subsetSum(int index, int currentSum, int size) {

if (currentSum == targetSum) {

printf("Subset: ");

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

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

}

printf("\n");

return;

}

if (currentSum > targetSum || index >= n) {

return;

}

solution[size] = set[index];

subsetSum(index + 1, currentSum + set[index], size + 1);

subsetSum(index + 1, currentSum, size);

}

int main() {

printf("Enter the number of elements in the set: ");

scanf("%d", &n);

printf("Enter the elements of the set:\n");

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

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

}

printf("Enter the target sum: ");

scanf("%d", &targetSum);

printf("Subsets with the sum %d:\n", targetSum);

subsetSum(0, 0, 0);

return 0;

}

#### OUTPUT:

## 3. Write a program to perform graph coloring problem using backtracking.

#include <stdio.h>

#include <stdbool.h>

#define MAX\_VERTICES 100

int graph[MAX\_VERTICES][MAX\_VERTICES];

int colors[MAX\_VERTICES];

int numVertices, numColors;

bool isSafe(int v, int c) {

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

if (graph[v][i] && colors[i] == c) {

return false;

}

}

return true;

}

bool graphColoring(int v) {

if (v == numVertices) {

return true;

}

for (int c = 1; c <= numColors; c++) {

if (isSafe(v, c)) {

colors[v] = c;

if (graphColoring(v + 1)) {

return true;

}

colors[v] = 0;

}

}

return false;

}

int main() {

printf("Enter the number of vertices: ");

scanf("%d", &numVertices);

printf("Enter the adjacency matrix:\n");

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

for (int j = 0; j < numVertices; j++) {

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

}

}

printf("Enter the number of colors: ");

scanf("%d", &numColors);

if (graphColoring(0)) {

printf("Graph can be colored using %d colors as follows:\n", numColors);

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

printf("Vertex %d: Color %d\n", i + 1, colors[i]);

}

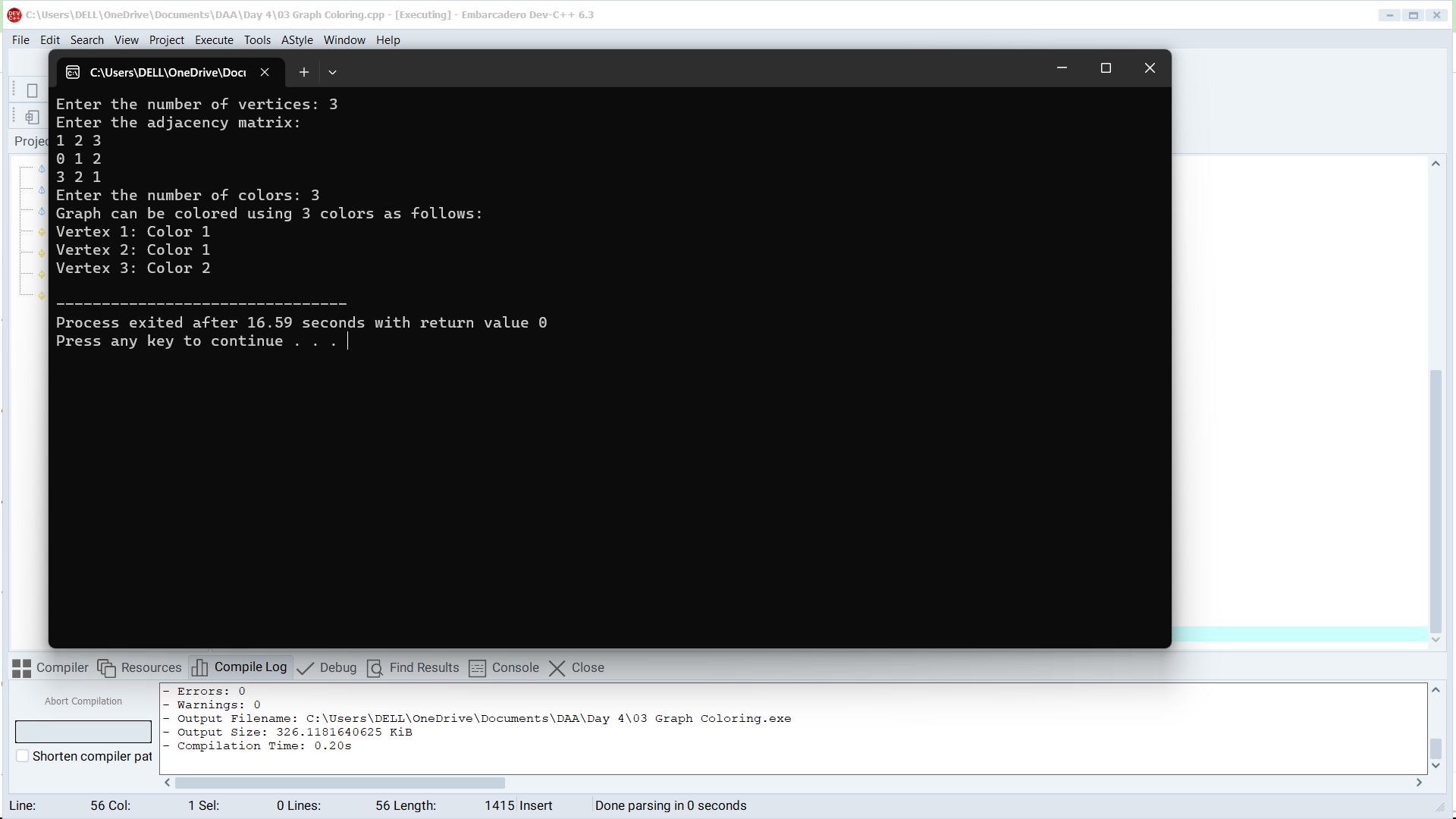
} else {

printf("Graph cannot be colored with %d colors.\n", numColors);

}

return 0;

}

OUTPUT:

## 4. Write a program to compute container loader Problem.

#include <stdio.h>

#include <stdbool.h>

#define MAX\_CONTAINERS 100

#define MAX\_ITEMS 100

int containers[MAX\_CONTAINERS];

int items[MAX\_ITEMS];

int numContainers, numItems;

void containerLoading() {

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

for (int j = i + 1; j < numItems; j++) {

if (items[i] < items[j]) {

int temp = items[i];

items[i] = items[j];

items[j] = temp;

}

}

}

int containerIndex = 0;

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

containers[i] = 0;

}

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

bool placed = false;

for (int j = 0; j <= containerIndex; j++) {

if (containers[j] + items[i] <= 100) {

containers[j] += items[i];

placed = true;

break;

}

}

if (!placed) {

containerIndex++;

containers[containerIndex] = items[i];

}

}

printf("Container Loading Result:\n");

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

printf("Container %d: %d/%d\n", i + 1, containers[i], 100);

}

}

int main() {

printf("Enter the number of containers: ");

scanf("%d", &numContainers);

printf("Enter the number of items: ");

scanf("%d", &numItems);

printf("Enter the sizes of items:\n");

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

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

}

containerLoading();

return 0;

}

#### OUTPUT:

### 5. Write a program to generate the list of all factor for n value.

#include <stdio.h>

int main() {

int n;

printf("Enter a positive integer: ");

scanf("%d", &n);

if (n <= 0) {

printf("Please enter a positive integer.\n");

} else {

printf("Factors of %d are: ", n);

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

if (n % i == 0) {

printf("%d ", i);

}

}

printf("\n");

}

return 0;

}

#### OUTPUT:

### 6. Write a program to perform Assignment problem using branch and bound.

#include <stdio.h>

#include <limits.h>

#define N 5

int costMatrix[N][N];

int assignment[N];

int usedRows[N], usedCols[N];

int minCost = INT\_MAX;

void printAssignment() {

printf("Assignment:\n");

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

printf("Agent %d is assigned to Task %d (Cost %d)\n", i + 1, assignment[i] + 1, costMatrix[i][assignment[i]]);

}

}

void branchAndBound(int agent, int costSoFar) {

if (agent == N) {

if (costSoFar < minCost) {

minCost = costSoFar;

printAssignment();

}

return;

}

for (int task = 0; task < N; task++) {

if (!usedCols[task] && (costSoFar + costMatrix[agent][task] < minCost)) {

assignment[agent] = task;

usedCols[task] = 1;

branchAndBound(agent + 1, costSoFar + costMatrix[agent][task]);

usedCols[task] = 0;

}

}

}

int main() {

printf("Enter the cost matrix (%d x %d):\n", N, N);

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

for (int j = 0; j < N; j++) {

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

}

}

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

assignment[i] = -1;

}

branchAndBound(0, 0);

printf("Minimum Cost: %d\n", minCost);

return 0;

}

#### OUTPUT:

### 7. Write a program for to perform liner search.

#include <stdio.h>

int linearSearch(int arr[], int size, int target) {

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

if (arr[i] == target) {

return i;

}

}

return -1;

}

int main() {

int arr[100];

int size, target;

printf("Enter the size of the array: ");

scanf("%d", &size);

printf("Enter %d elements for the array:\n", size);

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

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

}

printf("Enter the element you want to search for: ");

scanf("%d", &target);

int result = linearSearch(arr, size, target);

if (result != -1) {

printf("Element %d found at index %d.\n", target, result);

} else {

printf("Element %d not found in the array.\n", target);

}

return 0;

}

#### OUTPUT:

### 8. Write a program to find out Hamiltonian circuit Using backtracking method

#include <stdio.h>

#include <stdbool.h>

#define V 5

int path[V];

bool visited[V];

void printHamiltonianCircuit() {

printf("Hamiltonian Circuit: ");

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

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

}

printf("%d\n", path[0]);

}

bool isSafe(int v, int pos, int graph[V][V]) {

if (!graph[path[pos - 1]][v])

return false;

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

if (path[i] == v)

return false;

}

return true;

}

bool hamiltonianCircuitUtil(int graph[V][V], int pos) {

if (pos == V) {

if (graph[path[pos - 1]][path[0]] == 1) {

printHamiltonianCircuit();

return true;

}

return false;

}

for (int v = 1; v < V; v++) {

if (!visited[v] && isSafe(v, pos, graph)) {

path[pos] = v;

visited[v] = true;

if (hamiltonianCircuitUtil(graph, pos + 1))

return true;

path[pos] = -1;

visited[v] = false;

}

}

return false;

}

bool findHamiltonianCircuit(int graph[V][V]) {

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

path[i] = -1;

visited[i] = false;

}

path[0] = 0;

visited[0] = true;

if (hamiltonianCircuitUtil(graph, 1) == false) {

printf("No Hamiltonian Circuit exists\n");

return false;

}

return true;

}

int main() {

int graph[V][V];

printf("Enter the adjacency matrix (%d x %d) for the graph:\n", V, V);

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

for (int j = 0; j < V; j++) {

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

}

}

if (findHamiltonianCircuit(graph) == false)

printf("No Hamiltonian Circuit exists\n");

return 0;

}

#### OUTPUT: