

19.

```

4  }
5  }
6  void knapsackGreedy(int w, struct Item items[], int n) {
7      int i, j, currentWeight;
8      float totalValue = 0.0;
9      for (i = 0; i < n; i++) {
10         items[i].value = items[i].value / items[i].weight;
11     }
12     for (i = 0; i < n - 1; i++) {
13         for (j = i + 1; j < n; j++) {
14             if (items[i].value < items[j].value) {
15                 struct Item temp = items[i];
16                 items[i] = items[j];
17                 items[j] = temp;
18             }
19         }
20     }
21     for (i = 0; i < n; i++) {
22         if (items[i].weight <= w) {
23             totalValue += items[i].value;
24             w -= items[i].weight;
25         } else {
26             totalValue += items[i].value * ((float) w / items[i].weight);
27             break;
28         }
29     }
30     printf("Maximum value in Knapsack: %.2f\n", totalValue);
31 }
32
33 int main() {
34     int w = 50;
35     struct Item items[] = {{10, 60}, {20, 100}, {30, 120}};
36     int n = sizeof(items) / sizeof(items[0]);
37     knapsackGreedy(w, items, n);
38     return 0;
39 }

```

Maximum value in Knapsack: 13.67

Process exited after 1.161 seconds with return value 0

Press any key to continue . . .

20.

```

4  struct Edge {
5      int u, v, weight;
6  };
7
8  int compare(const Edge a, const Edge b) {
9      return a.weight < b.weight;
10 }
11
12 void printEdges(const Edge edges[], int n) {
13     for (int i = 0; i < n; i++) {
14         printf("%d - %d %d\n", edges[i].u, edges[i].v, edges[i].weight);
15     }
16 }
17
18 void printMST(const Edge edges[], int n) {
19     for (int i = 0; i < n; i++) {
20         printf("%d - %d %d\n", edges[i].u, edges[i].v, edges[i].weight);
21     }
22 }
23
24 int main() {
25     Edge edges[] = {{0, 1, 2}, {1, 2, 3}, {0, 3, 6}, {1, 4, 5}};
26     int n = sizeof(edges) / sizeof(edges[0]);
27     printEdges(edges, n);
28     printMST(edges, n);
29     return 0;
30 }

```

Edge Weight

0 - 1 2

1 - 2 3

0 - 3 6

1 - 4 5

Process exited after 1.093 seconds with return value 0

Press any key to continue . . .

21.

```

1 #include <stdio.h>
2 #include <limits.h>
3
4 int sur(int freq[], int l, int j) {
5     int s = 0;
6     for (int k = l; k <= j; k++)
7         s += freq[k];
8     return s;
9 }
10
11 int optimalBST(int keys[], int freq[], int n) {
12     int cost[n][n];
13     for (int l = 0; l < n; l++)
14         cost[l][l] = freq[l];
15
16     for (int l = 2; l <= n; l++) {
17         for (int i = 0; i <= n - l; i++) {
18             int j = i + l - 1;
19             cost[i][j] = INT_MAX;
20
21             for (int r = i; r <= j; r++) {
22                 int c = (i > 0 ? cost[i][r - 1] : 0) +
23                     ((r < j) ? cost[r + 1][j] : 0) +
24                     sur(freq, i, j);
25                 if (c < cost[i][j])
26                     cost[i][j] = c;
27             }
28         }
29     }
30     return cost[n - 1][n - 1];
31 }
32
33 int main() {
34     int keys[] = {10, 12, 20};
35     int freq[] = {34, 8, 56};
36     int n = sizeof(keys) / sizeof(keys[0]);
37     printf("Cost of optimal BST is: %d", optimalBST(keys, freq, n));
38     return 0;
39 }

```

Cost of optimal BST is: 142
 Process exited after 1.886 seconds with return value 0
 Press any key to continue . . .

22.

```

1 #include <stdio.h>
2
3 int binomialCoeff(int n, int k) {
4     int C[n + 1][k + 1];
5     int i, j;
6     for (i = 0; i <= n; i++) {
7         for (j = 0; j <= k; j++) {
8             if (j == 0 || j == i)
9                 C[i][j] = 1;
10            else
11                C[i][j] = C[i - 1][j - 1] + C[i - 1][j];
12        }
13    }
14    return C[n][k];
15 }
16
17 int main() {
18     int n = 5, k = 2;
19     printf("Binomial Coefficient C(%d, %d) is: %d", n, k, binomialCoeff(n, k));
20     return 0;
21 }

```

Binomial Coefficient C(5, 2) is: 10
 Process exited after 1.2 seconds with return value 0
 Press any key to continue . . .

23.

```

1 #include <stdio.h>
2
3 int main() {
4     int n, reverse = 0, remainder;
5
6     printf("Enter an integer: ");
7     scanf("%d", &n);
8
9
10    while (n != 0) {
11        remainder = n % 10;
12        reverse = reverse * 10 + remainder;
13        n /= 10;
14    }
15
16    printf("Reversed number = %d", reverse);
17
18    return 0;
19 }

```

Output Window:

```

Enter an integer: 12345
Reversed number = 54321
-----
Process exited after 4.859 seconds with return value 0
Press any key to continue . . .

```

Compiler Output:

```

-----
Errors: 0
Warnings: 0
Output Filename: C:\Users\nagaj\Documents\DA\EXP-24.exe
Output Size: 128.1015625 KIB
Compilation Time: 0.19s

```

24.

```

1 #include <stdio.h>
2
3 int isPerfectNumber(int num) {
4     int sum = 0;
5     for (int i = 1; i <= num / 2; i++) {
6         if (num % i == 0) {
7             sum += i;
8         }
9     }
10    return sum == num;
11 }
12
13 int main() {
14     int num = 28;
15     if (isPerfectNumber(num)) {
16         printf("%d is a perfect number.", num);
17     } else {
18         printf("%d is not a perfect number.", num);
19     }
20    return 0;
21 }

```

Output Window:

```

28 is a perfect number.
-----
Process exited after 1.912 seconds with return value 0
Press any key to continue . . .

```

Compiler Output:

```

-----
Errors: 0
Warnings: 0
Output Filename: C:\Users\nagaj\Documents\DA\EXP-25.exe
Output Size: 128.4697265625 KIB
Compilation Time: 0.20s

```

25.

```

1 #include <stdio.h>
2 #include <limits.h>
3
4 #define V 4
5
6 int tsp(int graph[V][V], int mask, int pos, int do){
7     if (mask == (1 << V) - 1)
8         return graph[pos][0];
9     if (do[pos][mask] != -1)
10         return do[pos][mask];
11     int minCost = INT_MAX;
12     for (int city = 0; city < V; city++) {
13         if ((mask & (1 << city)) == 0) {
14             int newCost = graph[pos][city] + tsp(graph, mask | (1 << city), city, do);
15             minCost = (newCost < minCost) ? newCost : minCost;
16         }
17     }
18     do[pos][mask] = minCost;
19     return do[pos][mask];
20 }
21
22 int main() {
23     int graph[V][V] = {
24         {0, 10, 15, 20},
25         {10, 0, 35, 25},
26         {15, 35, 0, 30},
27         {20, 25, 30, 0}
28     };
29     int do[V][1 << V];
30     for (int i = 0; i < V; i++)
31         for (int j = 0; j < (1 << V); j++)
32             do[i][j] = -1;
33     int minCost = tsp(graph, 1, 0, do);
34     printf("Minimum cost for the Travelling Salesman Problem is: %d", minCost);
35     return 0;
36 }

```

Minimum cost for the Travelling Salesman Problem is: 80

Process exited after 1.971 seconds with return value 0

Press any key to continue . . .

26.

```

1 #include <stdio.h>
2
3 void printPattern(int n) {
4     if (n == 0)
5         return;
6     printPattern(n - 1);
7     for (int i = 1; i <= n; i++) {
8         printf("%d ", i);
9     }
10    printf("\n");
11 }
12
13 int main() {
14     int rows = 4;
15     printPattern(rows);
16     return 0;
17 }

```

1
12
123
1234

Process exited after 3.07 seconds with return value 0

Press any key to continue . . .

27.

The screenshot shows a C++ IDE with a file named EXP-29.cpp. The code implements the Floyd-Warshall algorithm to find the shortest distances between every pair of vertices in a graph. The graph is represented as a 2D array 'graph' of size V x V, where V is the number of vertices. The algorithm uses three nested loops: the outermost loop iterates over the source vertex 'i', the middle loop iterates over the destination vertex 'j', and the innermost loop iterates over the intermediate vertex 'k'. The distance between 'i' and 'j' is updated if the path through 'k' is shorter. The output is a matrix of shortest distances.

```

1 #include <stdio.h>
2 #include <limits.h>
3
4 #define V 4
5
6 void floydwarshall(int graph[V][V]) {
7     int dist[V][V];
8     for (int i = 0; i < V; i++) {
9         for (int j = 0; j < V; j++) {
10             dist[i][j] = graph[i][j];
11         }
12     }
13
14     for (int k = 0; k < V; k++) {
15         for (int i = 0; i < V; i++) {
16             for (int j = 0; j < V; j++) {
17                 if (dist[i][k] != INT_MAX && dist[k][j] != INT_MAX && dist[i][k] + dist[k][j] < dist[i][j]) {
18                     dist[i][j] = dist[i][k] + dist[k][j];
19                 }
20             }
21         }
22     }
23
24     printf("Shortest distances between every pair of vertices:\n");
25     for (int i = 0; i < V; i++) {
26         for (int j = 0; j < V; j++) {
27             if (dist[i][j] == INT_MAX) {
28                 printf("INF\t");
29             } else {
30                 printf("%d\t", dist[i][j]);
31             }
32         }
33         printf("\n");
34     }
35 }
36
37 int main() {
38     int graph[V][V] = {
39         {0, 5, 8, 9},
40         {INT_MAX, 0, 3, 4},
41         {INT_MAX, INT_MAX, 0, 1},
42         {INT_MAX, INT_MAX, INT_MAX, 0}
43     };
44
45     floydwarshall(graph);
46
47     return 0;
48 }

```

The output window shows the following shortest distances between every pair of vertices:

0	5	8	9
INF	0	3	4
INF	INF	0	1
INF	INF	INF	0

Process exited after 0.7291 seconds with return value 0
Press any key to continue . . .

28.

The screenshot shows a C++ IDE with a file named EXP-30.cpp. The code implements a program that prints a Pascal's triangle. It takes the number of rows as input and prints the triangle using spaces and numbers. The output is a Pascal's triangle with 4 rows.

```

1 #include <stdio.h>
2 int main() {
3     int rows, coef = 1, space, i, j;
4     printf("Enter the number of rows: ");
5     scanf("%d", &rows);
6     for (i = 0; i < rows; i++) {
7         for (space = 0; space <= rows - i - 1; space++) {
8             printf(" ");
9         }
10        for (j = 0; j <= i; j++) {
11            if (j == 0 || i == 0) {
12                coef = 1;
13            } else {
14                coef = coef * (i - j + 1) / j;
15            }
16            printf("%d", coef);
17        }
18        printf("\n");
19    }
20    return 0;
21 }

```

The output window shows the following Pascal's triangle:

```

Enter the number of rows: 4
      1
     1 1
    1 2 1
   1 3 3 1

```

Process exited after 6.434 seconds with return value 0
Press any key to continue . . .

29.

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 struct Node {
5     int data;
6     struct Node* next;
7 };
8
9 void insert_number(struct Node** head, int value) {
10     struct Node* new_node = (struct Node*)malloc(sizeof(struct Node));
11     new_node->data = value;
12     new_node->next = *head;
13     *head = new_node;
14 }
15
16 void print_list(struct Node* head) {
17     struct Node* temp = head;
18     while (temp != NULL) {
19         printf("%d -> ", temp->data);
20         temp = temp->next;
21     }
22     printf("NULL\n");
23 }
24
25 int main() {
26     struct Node* head = NULL;
27     insert_number(&head, 5);
28     insert_number(&head, 10);
29     insert_number(&head, 15);
30     printf("List after insertion: ");
31     print_list(head);
32     return 0;
33 }

```

Output:

```

List after insertion: 15 -> 10 -> 5 -> NULL
Process exited after 1.385 seconds with return value 0
Press any key to continue . . .

```

30.

```

1 #include <stdio.h>
2
3 int sumOfDigits(int num) {
4     if (num == 0)
5         return 0;
6     return (num % 10) + sumOfDigits(num / 10);
7 }
8
9 int main() {
10     int number = 12345;
11     int sum = sumOfDigits(number);
12     printf("Sum of digits of %d is: %d\n", number, sum);
13     return 0;
14 }

```

Output:

```

Sum of digits of 12345 is: 15
Process exited after 1.637 seconds with return value 0
Press any key to continue . . .

```

31.

```

1 #include <iostream>
2 #include <stdio.h>
3
4 void findMinimumMaximum(int arr[], int n)
5 {
6     int i;
7
8     int minE = INT_MIN, maxE = INT_MAX;
9
10    for (i = 0; i < n; i++) {
11        if (arr[i] < minE) {
12            minE = arr[i];
13        }
14        if (arr[i] > maxE) {
15            maxE = arr[i];
16        }
17    }
18
19    printf("The minimum element is %d", minE);
20    printf("\n");
21    printf("The maximum element is %d", maxE);
22
23    return;
24 }
25
26 int main()
27 {
28     int arr[] = { 1, 2, 4, -1 };
29
30     int n = sizeof(arr) / sizeof(arr[0]);
31
32     findMinimumMaximum(arr, n);
33
34     return 0;
35 }

```

Output: The minimum element is -1
The maximum element is 4
Process exited after 0.8437 seconds with return value 0
Press any key to continue . . .

32.

```

1 #include <iostream>
2 #include <stdio.h>
3
4 void printSolution(int board[3][3]) {
5     for (int i = 0; i < 3; i++) {
6         for (int j = 0; j < 3; j++) {
7             printf("%d ", board[i][j]);
8         }
9         printf("\n");
10    }
11 }
12
13 bool isSafe(int board[3][3], int row, int col) {
14     int i, j;
15     for (i = 0; i < 3; i++) {
16         if (board[i][col] != 0) {
17             return false;
18         }
19     }
20     for (j = 0; j < 3; j++) {
21         if (board[row][j] != 0) {
22             return false;
23         }
24     }
25     int sum = 0;
26     for (i = 0; i < 3; i++) {
27         sum += board[i][col];
28     }
29     for (j = 0; j < 3; j++) {
30         sum += board[row][j];
31     }
32     if (sum != 15) {
33         return false;
34     }
35     return true;
36 }
37
38 bool isSolvable(int board[3][3]) {
39     if (isSafe(board, 0, 0)) {
40         return true;
41     }
42     for (int i = 0; i < 3; i++) {
43         for (int j = 0; j < 3; j++) {
44             if (board[i][j] != 0) {
45                 board[i][j] = 0;
46                 if (isSolvable(board)) {
47                     return true;
48                 }
49                 board[i][j] = 0;
50             }
51         }
52     }
53     return false;
54 }
55
56 bool isSolvable(int board[3][3]) {
57     int board[3][3] = {{0, 0, 0},
58                         {0, 0, 0},
59                         {0, 0, 0}};
60
61     if (isSolvable(board)) {
62         printSolution(board);
63         return true;
64     }
65     return false;
66 }
67
68 int main() {
69     isSolvable();
70     return 0;
71 }

```

Output: 0 0 1 0
1 0 0 0
0 0 0 1
0 1 0 0
Process exited after 0.7386 seconds with return value 0
Press any key to continue . . .

33.

```

1 #include <iostream>
2
3 int isSubsetSum(int set[], int n, int sum) {
4     if (sum == 0) return 1;
5     if (n == 0) return 0;
6     if (set[n-1] > sum) return isSubsetSum(set, n-1, sum);
7     return isSubsetSum(set, n-1, sum) || isSubsetSum(set, n-1, sum - set[n-1]);
8 }
9
10 int main() {
11     int set[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
12     int n = sizeof(set) / sizeof(set[0]);
13     int sum = 15;
14     if (isSubsetSum(set, n, sum)) printf("Found a subset with given sum");
15     else printf("No subset with given sum");
16     return 0;
17 }

```

Output: Found a subset with given sum

34.

```

1 #include <iostream>
2
3 int isSubsetSum(int set[], int n, int sum) {
4     if (sum == 0) return 1;
5     if (n == 0) return 0;
6     if (set[n-1] > sum) return isSubsetSum(set, n-1, sum);
7     return isSubsetSum(set, n-1, sum) || isSubsetSum(set, n-1, sum - set[n-1]);
8 }
9
10 int main() {
11     int set[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
12     int n = sizeof(set) / sizeof(set[0]);
13     int sum = 15;
14     if (isSubsetSum(set, n, sum)) printf("Found a subset with given sum");
15     else printf("No subset with given sum");
16     return 0;
17 }

```

Output: Found a subset with given sum

[illegible]

The image shows a C++ IDE with a source code editor on the left and an output window on the right. The source code defines a `container_loader` struct and a `main` function that interacts with the user to create a container, push elements, and check capacity. The output window shows the program's execution, including prompts for the number of containers, weights, and capacity, and the final output of the container's state.

```

4  int container_loader {0};
5  int stack[MAX_CONTAINERS];
6  int top = -1;
7  int n, capacity, current_weight = 0;
8
9  void container_loader {int k} {
10     if (k == 0) {
11         printf("Welcome!\n");
12         for (int i = 0; i <= top; i++) {
13             printf("%d", stack[i]);
14         }
15         printf("\n");
16         return;
17     }
18     if (current_weight + containers[k] >= capacity) {
19         current_weight = containers[k];
20         stack[top] = containers[k];
21         container_loader(k + 1);
22         current_weight = containers[k];
23         top++;
24     }
25     container_loader(k + 1);
26 }
27
28 int main() {
29     printf("Enter the number of containers: ");
30     scanf("%d", &n);
31     printf("Enter the weights of the containers:\n");
32     for (int i = 0; i <= n; i++) {
33         scanf("%d", &containers[i]);
34     }
35     printf("Enter the capacity of the loader: ");
36     scanf("%d", &capacity);
37     container_loader(0);
38     return 0;
39 }

```

The output window shows the program's execution, including prompts for the number of containers, weights, and capacity, and the final output of the container's state.

```

C:\Users\nagat\Documents\DA\EXP-37.cpp
Enter the number of containers: 4
Enter the weights of the containers:
2
2
4
9
Enter the capacity of the loader: 2
Solution: 2
Solution: 2
Solution:
Process exited after 60.34 seconds with return value 0
Press any key to continue . . .

```

37.

```

1 #include <stdio.h>
2
3 void generate_factors(int n, int i) {
4     if (i > n)
5         return;
6     if (n % i == 0) {
7         printf("%d ", i);
8     }
9     generate_factors(n, i + 1);
10 }
11
12 int main() {
13     int n;
14     printf("Enter the value of n: ");
15     scanf("%d", &n);
16     printf("Factors of %d are: ", n);
17     generate_factors(n, 1);
18     return 0;
19 }

```

Output:

```

Enter the value of n: 5
Factors of 5 are: 1 5
Process exited after 6.583 seconds with return value 0
Press any key to continue . . .

```

Compiler: GCC 4.9.2 64-bit Release
 Errors: 0
 Warnings: 0
 Output Filename: C:\Users\naga\Documents\DAI\EXP-38.exe
 Output Size: 128,641,601,542 B
 Compilation Time: 0.17s

38.

```

1 #include <stdio.h>
2 #include <limits.h>
3
4 #define N 4
5
6 int cost[N][N] = {
7     {0, 2, 4, 5},
8     {3, 15, 7, 12},
9     {5, 8, 4, 1},
10    {4, 7, 2, 8}
11 };
12
13 int min_cost = INT_MAX;
14 int assigned[N];
15 int visited[N] = {0};
16
17 void assign_task(int worker, int total_cost) {
18     if (total_cost < min_cost) {
19         min_cost = total_cost;
20         for (int i = 0; i < N; i++) {
21             assigned[i] = visited[i];
22         }
23     }
24     return;
25 }
26
27 for (int task = 0; task < N; task++) {
28     if (!visited[task]) {
29         visited[task] = 1;
30         assign_task(worker + 1, total_cost + cost[worker][task]);
31         visited[task] = 0;
32     }
33 }
34
35 int main() {
36     assign_task(0, 0);
37     printf("Minimum Cost: %d", min_cost);
38     printf("Assignment: ");
39     for (int i = 0; i < N; i++) {
40         printf("Worker %d -> Task %d, ", i + 1, assigned[i] + 1);
41     }
42     printf("\n");
43     return 0;
44 }

```

Output:

```

Minimum Cost: 8
Assignment: Worker 1 -> Task 2, Worker 2 -> Task 2, Worker 3 -> Task 2, Worker 4 -> Task 2,
Process exited after 0.5249 seconds with return value 0
Press any key to continue . . .

```

Line: 45, Col: 18, Sel: 0, Lines: 48, Length: 1010, Insert, Done parsing in 0 seconds

39.

```

1 #include <stdio.h>
2
3 int linearSearch(int arr[], int n, int key) {
4     for (int i = 0; i < n; i++) {
5         if (arr[i] == key) {
6             return i;
7         }
8     }
9     return -1;
10 }
11
12 int main() {
13     int arr[] = {10, 20, 30, 40, 50};
14     int key = 30;
15     int n = sizeof(arr) / sizeof(arr[0]);
16     int result = linearSearch(arr, n, key);
17     if (result != -1) {
18         printf("Element found at index: %d\n", result);
19     } else {
20         printf("Element not found\n");
21     }
22     return 0;
23 }

```

Output:

```

Element found at index: 2
-----
Process exited after 0.7782 seconds with return value 0
Press any key to continue . . .

```

Compiler Output:

```

-----
Errors: 0
Warnings: 0
Output Filename: C:\Users\nagaj\Documents\DA\EXP-32.exe
Output Size: 128.639871875 KiB
Compilation Time: 0.00s

```

40.

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 struct Node {
5     int data;
6     struct Node* next;
7 };
8
9 void insert_number(struct Node** head, int value) {
10     struct Node* new_node = (struct Node*)malloc(sizeof(struct Node));
11     new_node->data = value;
12     new_node->next = *head;
13     *head = new_node;
14 }
15
16 void print_list(struct Node* head) {
17     struct Node* temp = head;
18     while (temp != NULL) {
19         printf("%d -> ", temp->data);
20         temp = temp->next;
21     }
22     printf("NULL\n");
23 }
24
25 int main() {
26     struct Node* head = NULL;
27
28     insert_number(&head, 5);
29     insert_number(&head, 10);
30     insert_number(&head, 15);
31
32     printf("List after insertion: ");
33     print_list(head);
34
35     return 0;
36 }

```

Output:

```

List after insertion: 15 -> 10 -> 5 -> NULL
-----
Process exited after 1.872 seconds with return value 0
Press any key to continue . . .

```

Compiler Output:

```

-----
Errors: 0
Warnings: 0
Output Filename: C:\Users\nagaj\Documents\DA\EXP-40.exe
Output Size: 129.681640625 KiB
Compilation Time: 0.17s

```