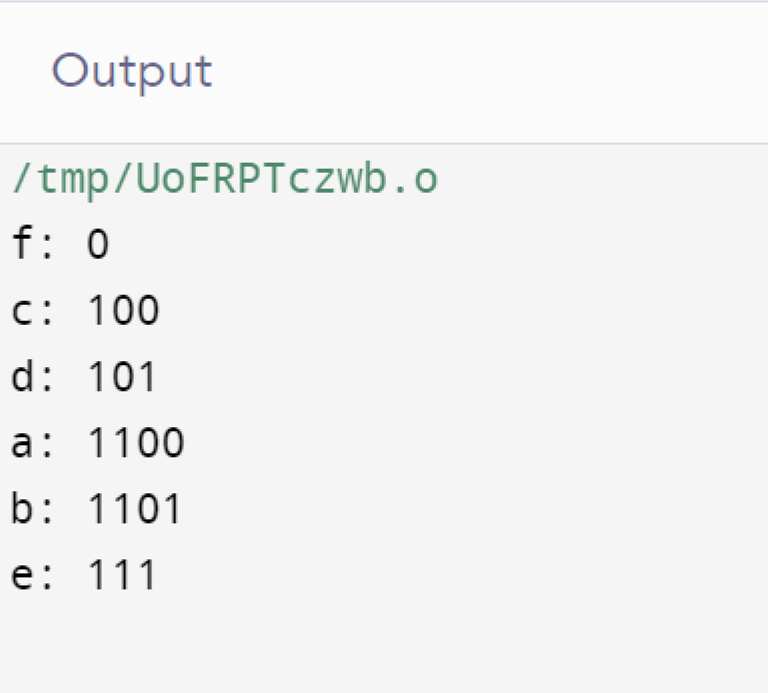
CODE:

```
// C++ program for Huffman Coding
#include <cstdlib>
#include <iostream>
using namespace std;

// This constant can be avoided by explicitly
// calculating height of Huffman Tree
#define MAX_TREE_HT 100

// A Huffman tree node
struct MinHeapNode {

        // One of the input characters
        char data;

        // Frequency of the character
        unsigned freq;

        // Left and right child of this node
        struct MinHeapNode *left, *right;
};

// A Min Heap: Collection of
// min-heap (or Huffman tree) nodes
struct MinHeap {

        // Current size of min heap
        unsigned size;

        // capacity of min heap
        unsigned capacity;

        // Array of minheap node pointers
        struct MinHeapNode** array;
};

// A utility function allocate a new
// min heap node with given character
// and frequency of the character
struct MinHeapNode* newNode(char data, unsigned freq)
{
        struct MinHeapNode* temp = (struct MinHeapNode*)malloc(
                sizeof(struct MinHeapNode));
```

```
    temp->left = temp->right = NULL;
    temp->data = data;
    temp->freq = freq;

    return temp;
}

// A utility function to create
// a min heap of given capacity
struct MinHeap* createMinHeap(unsigned capacity)

{

    struct MinHeap* minHeap
        = (struct MinHeap*)malloc(sizeof(struct MinHeap));

    // current size is 0
    minHeap->size = 0;

    minHeap->capacity = capacity;

    minHeap->array = (struct MinHeapNode**)malloc(
        minHeap->capacity * sizeof(struct MinHeapNode*));
    return minHeap;
}

// A utility function to
// swap two min heap nodes
void swapMinHeapNode(struct MinHeapNode** a,
                    struct MinHeapNode** b)

{

    struct MinHeapNode* t = *a;
    *a = *b;
    *b = t;
}

// The standard minHeapify function.
void minHeapify(struct MinHeap* minHeap, int idx)

{
    int smallest = idx;
    int left = 2 * idx + 1;
    int right = 2 * idx + 2;

    if (left < minHeap->size
        && minHeap->array[left]->freq
            < minHeap->array[smallest]->freq)
        smallest = left;

    if (right < minHeap->size
        && minHeap->array[right]->freq
            < minHeap->array[smallest]->freq)
        smallest = right;

    if (smallest != idx) {
        swapMinHeapNode(&minHeap->array[smallest],
                        &minHeap->array[idx]);
        minHeapify(minHeap, smallest);
    }
}

// A utility function to check
// if size of heap is 1 or not
int isSizeOne(struct MinHeap* minHeap)
{

    return (minHeap->size == 1);
}

// A standard function to extract
// minimum value node from heap
struct MinHeapNode* extractMin(struct MinHeap* minHeap)

{

    struct MinHeapNode* temp = minHeap->array[0];
    minHeap->array[0] = minHeap->array[minHeap->size - 1];

    --minHeap->size;
    minHeapify(minHeap, 0);

    return temp;
}
```

```
// A utility function to insert
// a new node to Min Heap
void insertMinHeap(struct MinHeap* minHeap,
                                        struct MinHeapNode* minHeapNode)

{

          ++minHeap->size;
          int i = minHeap->size - 1;

          while (i
                    && minHeapNode->freq
                                        < minHeap->array[(i - 1) / 2]->freq) {

                    minHeap->array[i] = minHeap->array[(i - 1) / 2];
                    i = (i - 1) / 2;
          }

          minHeap->array[i] = minHeapNode;
}

// A standard function to build min heap
void buildMinHeap(struct MinHeap* minHeap)

{

          int n = minHeap->size - 1;
          int i;

          for (i = (n - 1) / 2; i >= 0; --i)
                    minHeapify(minHeap, i);
}

// A utility function to print an array of size n
void printArr(int arr[], int n)
{
          int i;
          for (i = 0; i < n; ++i)
                    cout << arr[i];

          cout << "\n";
}

// Utility function to check if this node is leaf
```

```
int isLeaf(struct MinHeapNode* root)

{

        return !(root->left) && !(root->right);
}

// Creates a min heap of capacity
// equal to size and inserts all character of
// data[] in min heap. Initially size of
// min heap is equal to capacity
struct MinHeap* createAndBuildMinHeap(char data[],
                                                                        int freq[], int size)

{

        struct MinHeap* minHeap = createMinHeap(size);

        for (int i = 0; i < size; ++i)
                minHeap->array[i] = newNode(data[i], freq[i]);

        minHeap->size = size;
        buildMinHeap(minHeap);

        return minHeap;
}

// The main function that builds Huffman tree
struct MinHeapNode* buildHuffmanTree(char data[],
                                                                        int freq[], int size)

{
        struct MinHeapNode *left, *right, *top;

        // Step 1: Create a min heap of capacity
        // equal to size. Initially, there are
        // modes equal to size.
        struct MinHeap* minHeap
                = createAndBuildMinHeap(data, freq, size);

        // Iterate while size of heap doesn't become 1
        while (!isSizeOne(minHeap)) {

                // Step 2: Extract the two minimum
                // freq items from min heap
                left = extractMin(minHeap);
                right = extractMin(minHeap);

                // Step 3: Create a new internal
                // node with frequency equal to the
                // sum of the two nodes frequencies.
                // Make the two extracted node as
                // left and right children of this new node.
                // Add this node to the min heap
                // '$' is a special value for internal nodes, not
                // used
                top = newNode('$', left->freq + right->freq);

                top->left = left;
                top->right = right;

                insertMinHeap(minHeap, top);
        }

        // Step 4: The remaining node is the
        // root node and the tree is complete.
        return extractMin(minHeap);
}

// Prints huffman codes from the root of Huffman Tree.
// It uses arr[] to store codes
void printCodes(struct MinHeapNode* root, int arr[],
                                int top)

{

        // Assign 0 to left edge and recur
        if (root->left) {

                arr[top] = 0;
                printCodes(root->left, arr, top + 1);
        }

        // Assign 1 to right edge and recur
        if (root->right) {

                arr[top] = 1;
                printCodes(root->right, arr, top + 1);
        }

        // If this is a leaf node, then
        // it contains one of the input
        // characters, print the character
        // and its code from arr[]
        if (isLeaf(root)) {

                cout << root->data << ": ";
                printArr(arr, top);
        }
}

// The main function that builds a
// Huffman Tree and print codes by traversing
// the built Huffman Tree
void HuffmanCodes(char data[], int freq[], int size)

{
        // Construct Huffman Tree
        struct MinHeapNode* root
                = buildHuffmanTree(data, freq, size);

        // Print Huffman codes using
        // the Huffman tree built above
        int arr[MAX_TREE_HT], top = 0;

        printCodes(root, arr, top);
}

// Driver code
int main()
{

        char arr[] = { 'a', 'b', 'c', 'd', 'e', 'f' };
        int freq[] = { 5, 9, 12, 13, 16, 45 };

        int size = sizeof(arr) / sizeof(arr[0]);

        HuffmanCodes(arr, freq, size);

        return 0;
}
```



```
/tmp/UoFRPTczwb.o
f: 0
c: 100
d: 101
a: 1100
b: 1101
e: 111
```