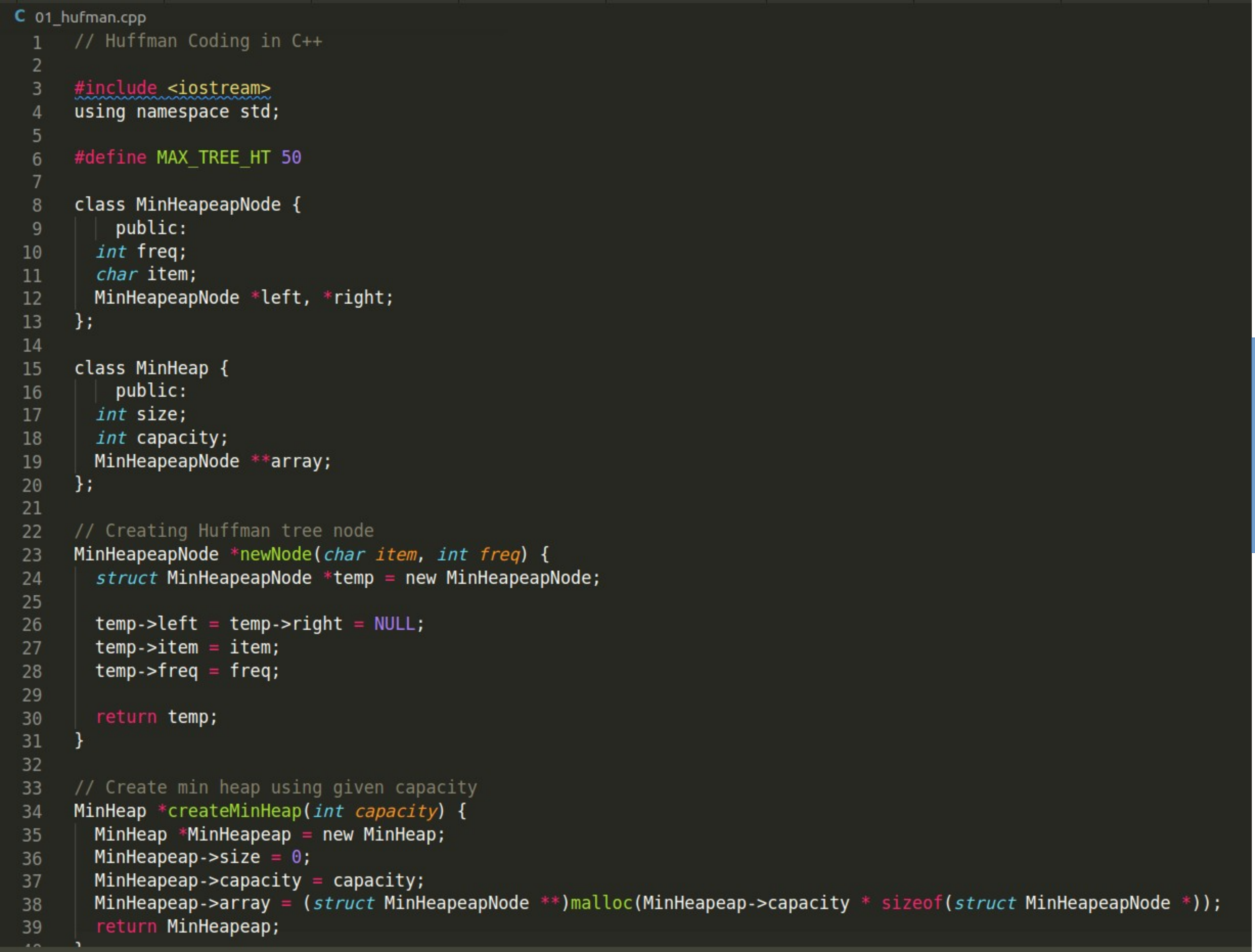
Name: **Kashif Ali**
Roll No: **20P-0648**
Section: **3D**

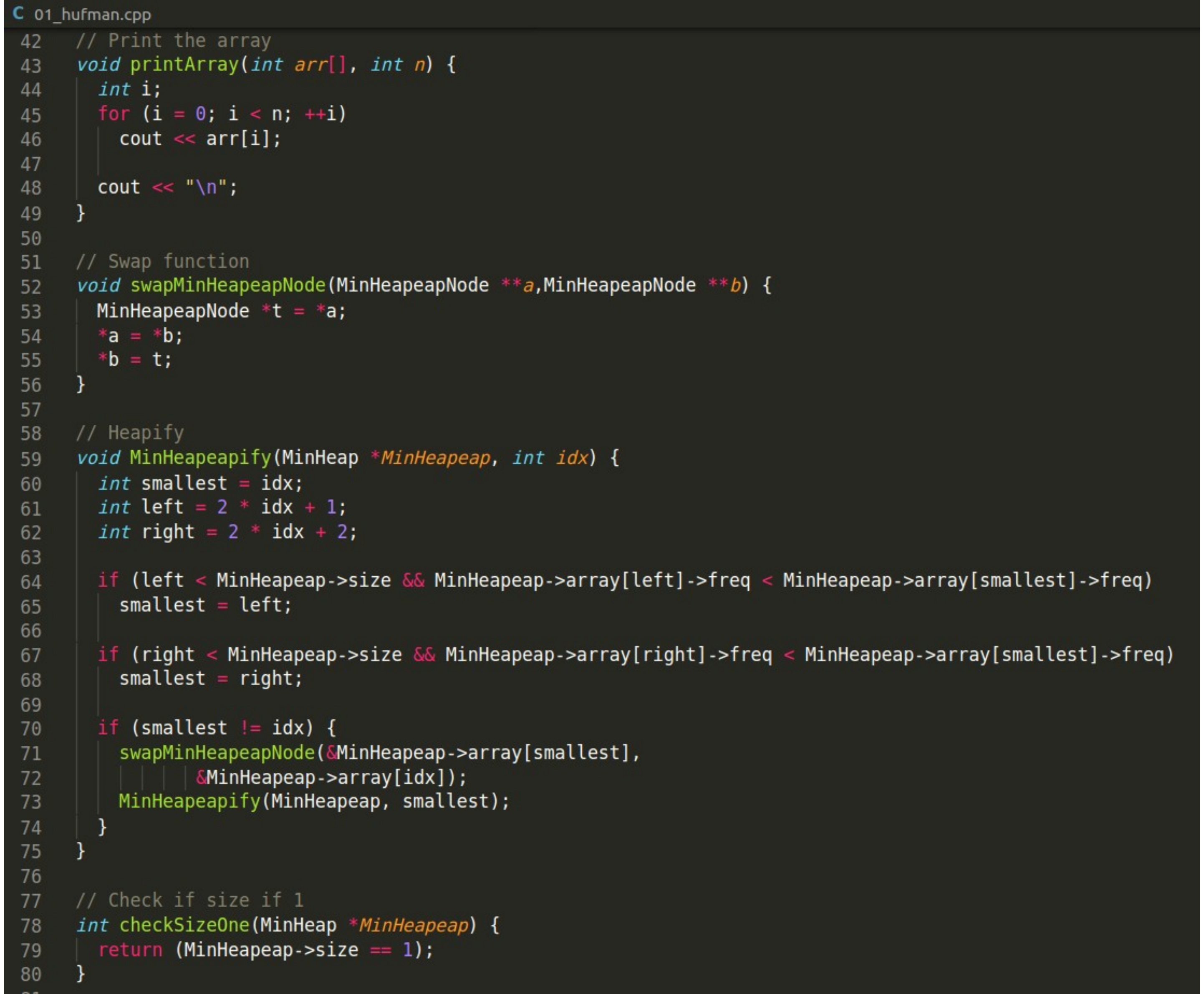
Lab-9 tasks
Huffman Coding



01_hufman.cpp

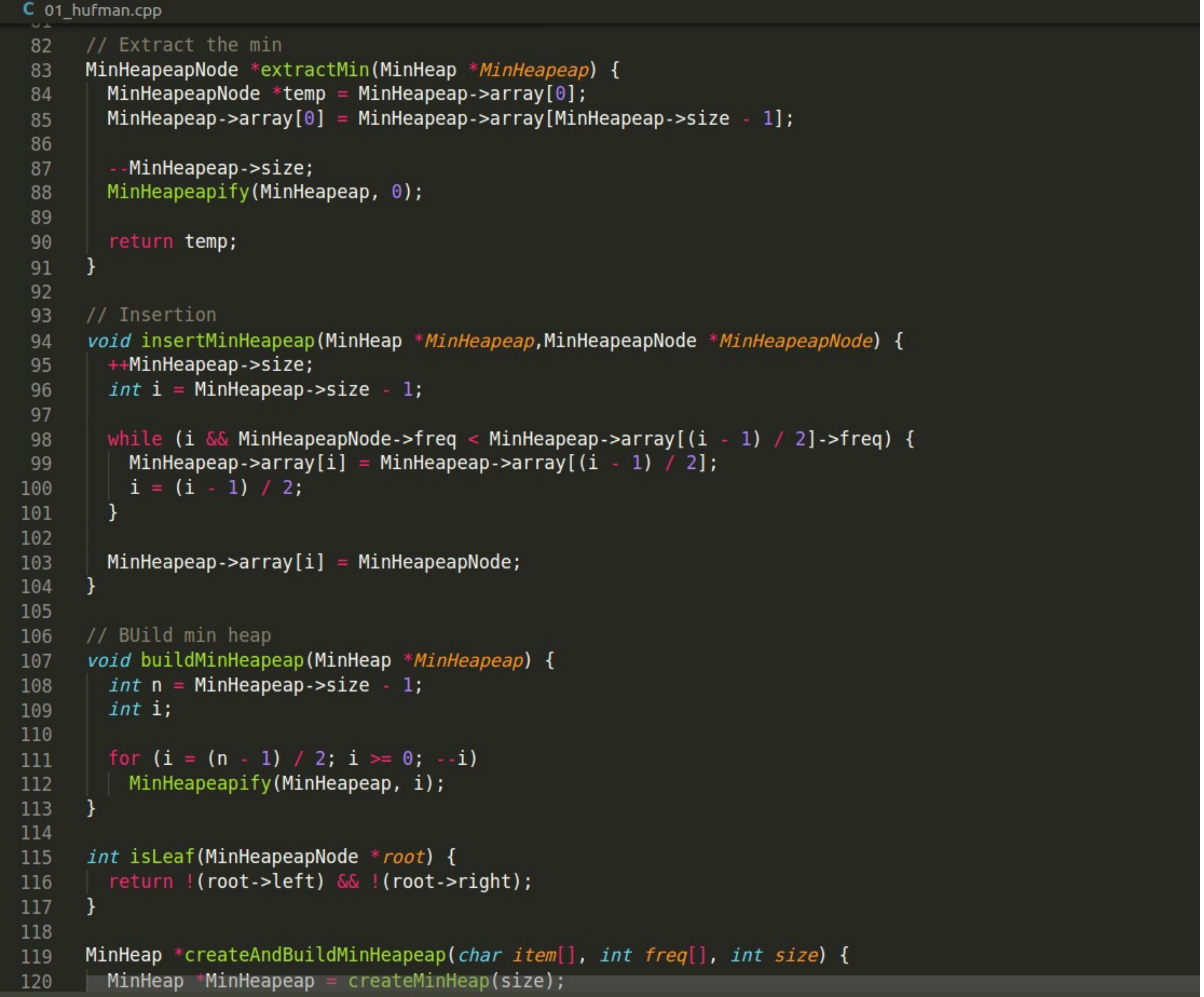
```cpp
// Huffman Coding in C++

#include <iostream>
using namespace std;

#define MAX_TREE_HT 50

class MinHeapeapNode {
    public:
  int freq;
  char item;
  MinHeapeapNode *left, *right;
};

class MinHeap {
    public:
  int size;
  int capacity;
  MinHeapeapNode **array;
};

// Creating Huffman tree node
MinHeapeapNode *newNode(char item, int freq) {
  struct MinHeapeapNode *temp = new MinHeapeapNode;

  temp->left = temp->right = NULL;
  temp->item = item;
  temp->freq = freq;

  return temp;
}

// Create min heap using given capacity
MinHeap *createMinHeap(int capacity) {
  MinHeap *MinHeapeap = new MinHeap;
  MinHeapeap->size = 0;
  MinHeapeap->capacity = capacity;
  MinHeapeap->array = (struct MinHeapeapNode **)malloc(MinHeapeap->capacity * sizeof(struct MinHeapeapNode *));
  return MinHeapeap;
```

Task-1
Huffman coding
part-a


01_hufman.cpp

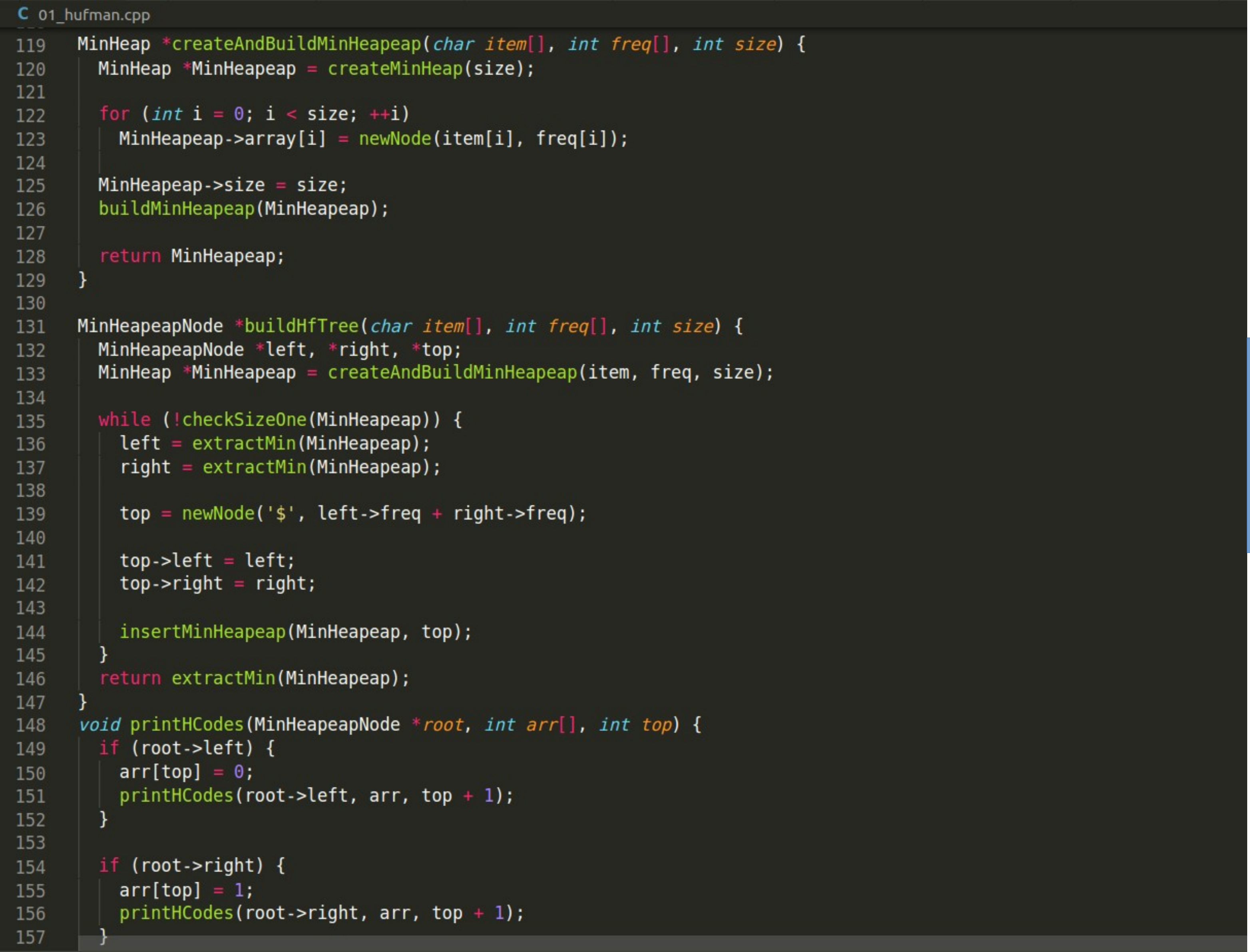
```cpp
// Print the array
void printArray(int arr[], int n) {
  int i;
  for (i = 0; i < n; ++i)
    cout << arr[i];

  cout << "\n";
}

// Swap function
void swapMinHeapeapNode(MinHeapeapNode **a,MinHeapeapNode **b) {
  MinHeapeapNode *t = *a;
  *a = *b;
  *b = t;
}

// Heapify
void MinHeapeapify(MinHeap *MinHeapeap, int idx) {
  int smallest = idx;
  int left = 2 * idx + 1;
  int right = 2 * idx + 2;

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

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

  if (smallest != idx) {
    swapMinHeapeapNode(&MinHeapeap->array[smallest],
           &MinHeapeap->array[idx]);
    MinHeapeapify(MinHeapeap, smallest);
  }
}

// Check if size if 1
int checkSizeOne(MinHeap *MinHeapeap) {
  return (MinHeapeap->size == 1);
}
```

Task-1
Huffman coding
part-b


01_hufman.cpp

```cpp
// Extract the min
MinHeapeapNode *extractMin(MinHeap *MinHeapeap) {
  MinHeapeapNode *temp = MinHeapeap->array[0];
  MinHeapeap->array[0] = MinHeapeap->array[MinHeapeap->size - 1];

  --MinHeapeap->size;
  MinHeapeapify(MinHeapeap, 0);

  return temp;
}

// Insertion
void insertMinHeapeap(MinHeap *MinHeapeap,MinHeapeapNode *MinHeapeapNode) {
  ++MinHeapeap->size;
  int i = MinHeapeap->size - 1;

  while (i && MinHeapeapNode->freq < MinHeapeap->array[(i - 1) / 2]->freq) {
    MinHeapeap->array[i] = MinHeapeap->array[(i - 1) / 2];
    i = (i - 1) / 2;
  }

  MinHeapeap->array[i] = MinHeapeapNode;
}

// BUild min heap
void buildMinHeapeap(MinHeap *MinHeapeap) {
  int n = MinHeapeap->size - 1;
  int i;

  for (i = (n - 1) / 2; i >= 0; --i)
    MinHeapeapify(MinHeapeap, i);
}

int isLeaf(MinHeapeapNode *root) {
  return !(root->left) && !(root->right);
}

MinHeap *createAndBuildMinHeapeap(char item[], int freq[], int size) {
  MinHeap *MinHeapeap = createMinHeap(size);
```

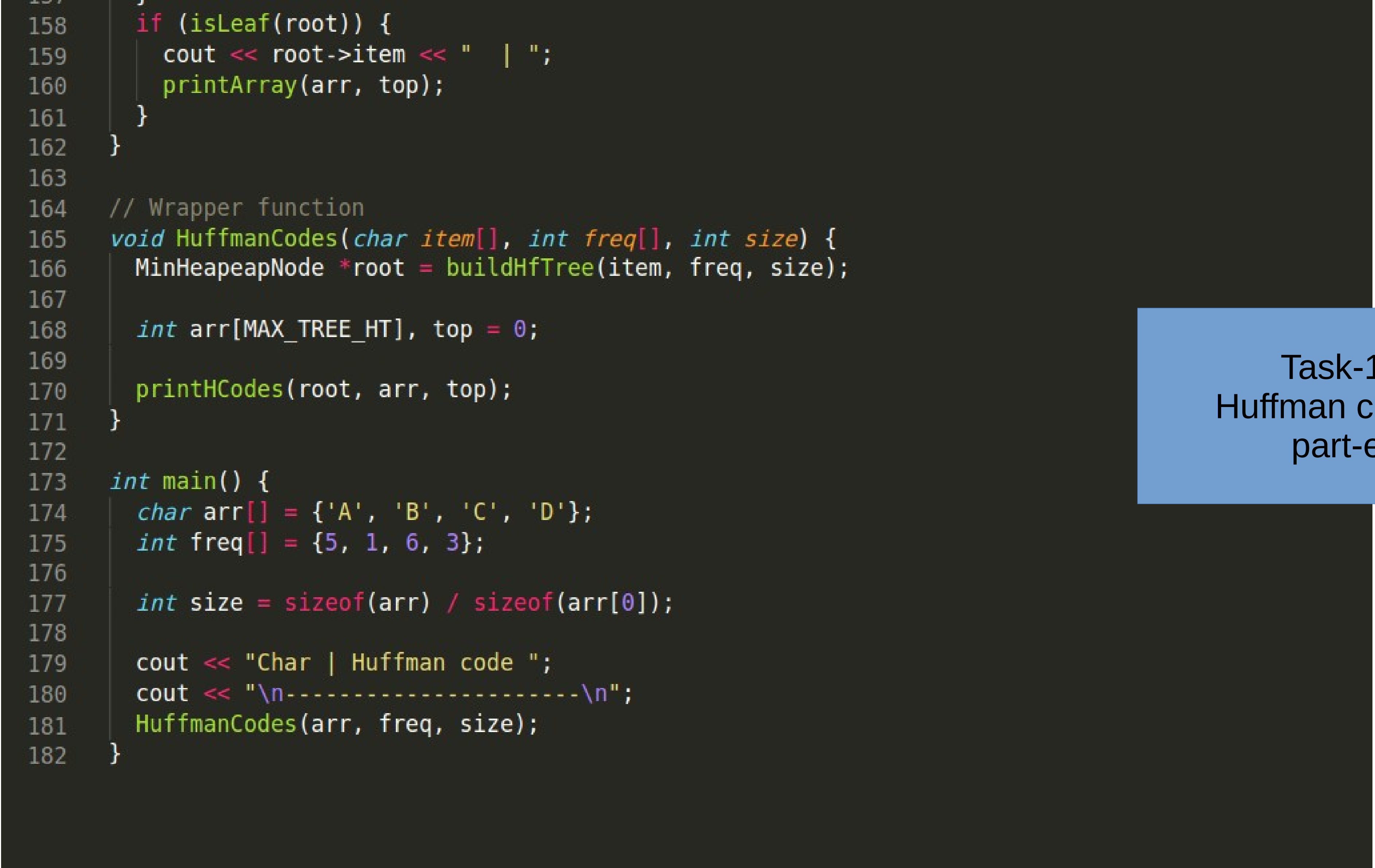
Task-1
Huffman coding
part-c



01_hufman.cpp

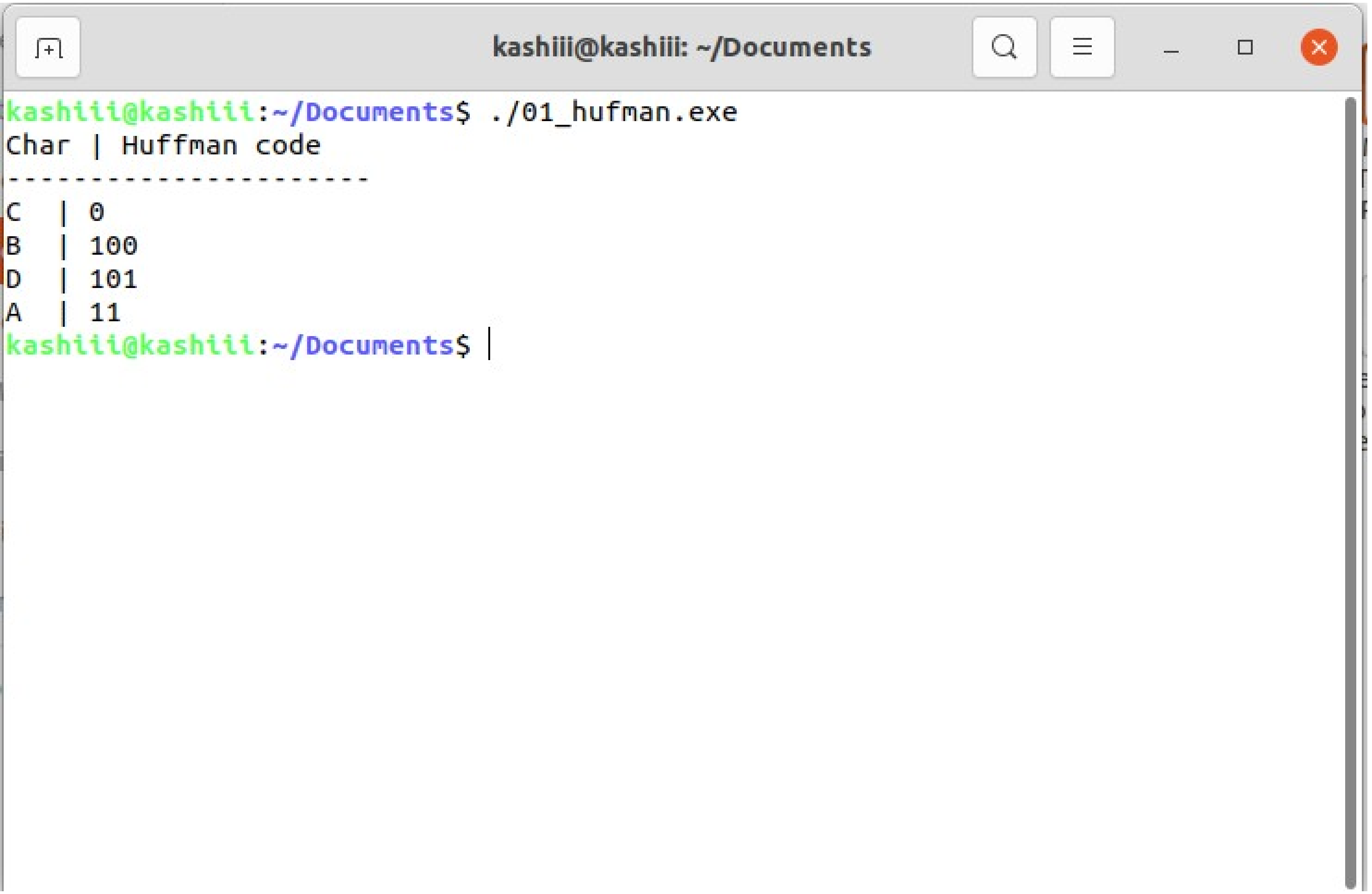
```cpp
MinHeap *createAndBuildMinHeapeap(char item[], int freq[], int size) {
  MinHeap *MinHeapeap = createMinHeap(size);

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

  MinHeapeap->size = size;
  buildMinHeapeap(MinHeapeap);

  return MinHeapeap;
}

MinHeapeapNode *buildHfTree(char item[], int freq[], int size) {
  MinHeapeapNode *left, *right, *top;
  MinHeap *MinHeapeap = createAndBuildMinHeapeap(item, freq, size);

  while (!checkSizeOne(MinHeapeap)) {
    left = extractMin(MinHeapeap);
    right = extractMin(MinHeapeap);

    top = newNode('$', left->freq + right->freq);

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

    insertMinHeapeap(MinHeapeap, top);
  }
  return extractMin(MinHeapeap);
}
void printHCodes(MinHeapeapNode *root, int arr[], int top) {
  if (root->left) {
    arr[top] = 0;
    printHCodes(root->left, arr, top + 1);
  }

  if (root->right) {
    arr[top] = 1;
    printHCodes(root->right, arr, top + 1);
  }
```

Task-1
Huffman coding
part-d



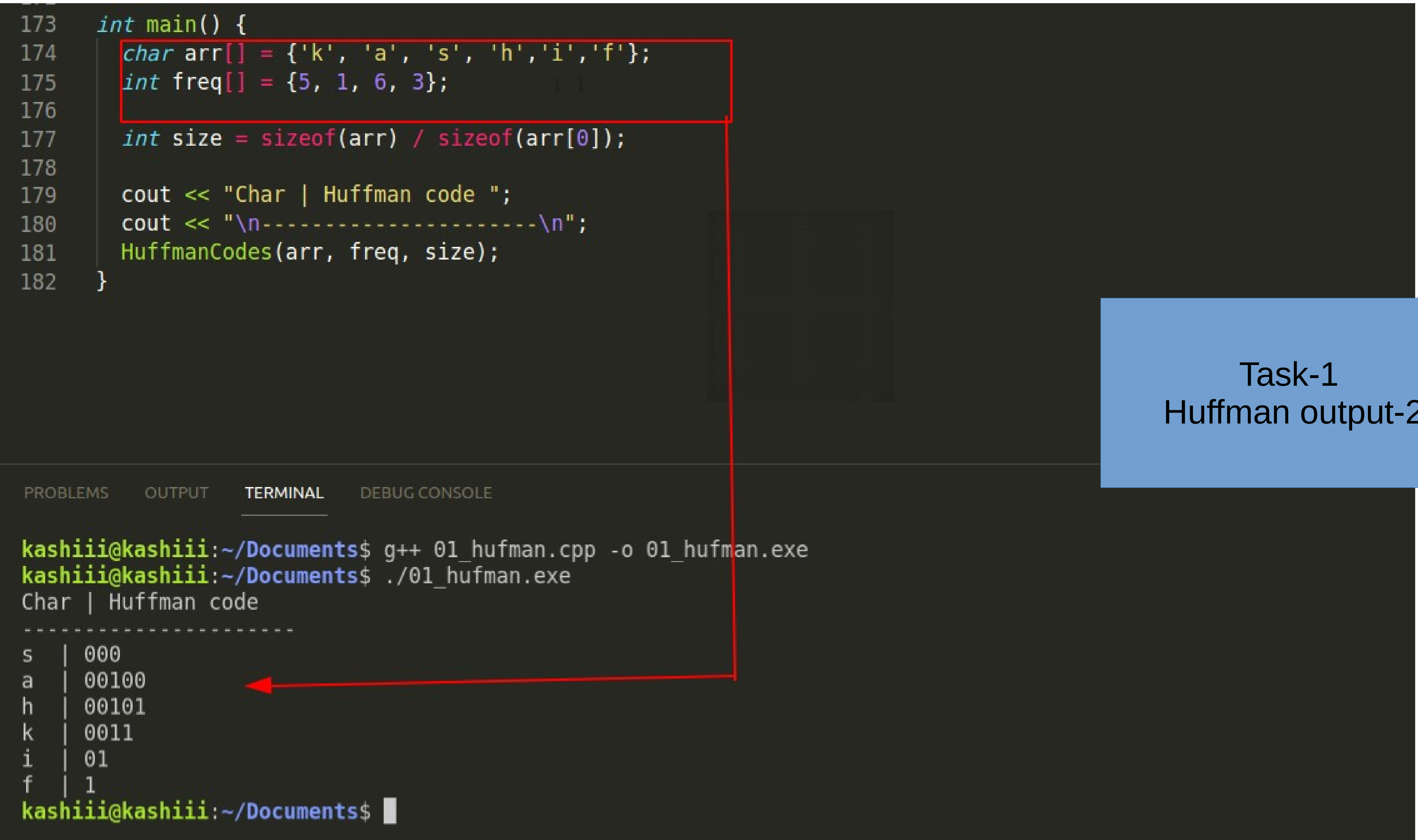
```cpp
  if (isLeaf(root)) {
    cout << root->item << "  | ";
    printArray(arr, top);
  }
}

// Wrapper function
void HuffmanCodes(char item[], int freq[], int size) {
  MinHeapeapNode *root = buildHfTree(item, freq, size);

  int arr[MAX_TREE_HT], top = 0;

  printHCodes(root, arr, top);
}

int main() {
  char arr[] = {'A', 'B', 'C', 'D'};
  int freq[] = {5, 1, 6, 3};

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

  cout << "Char | Huffman code ";
  cout << "\n---------------------\n";
  HuffmanCodes(arr, freq, size);
}
```

Task-1
Huffman coding
part-e



```
kashiii@kashiii:~/Documents$ ./01_hufman.exe
Char | Huffman code
----------------------
C  | 0
B  | 100
D  | 101
A  | 11
kashiii@kashiii:~/Documents$ 
```

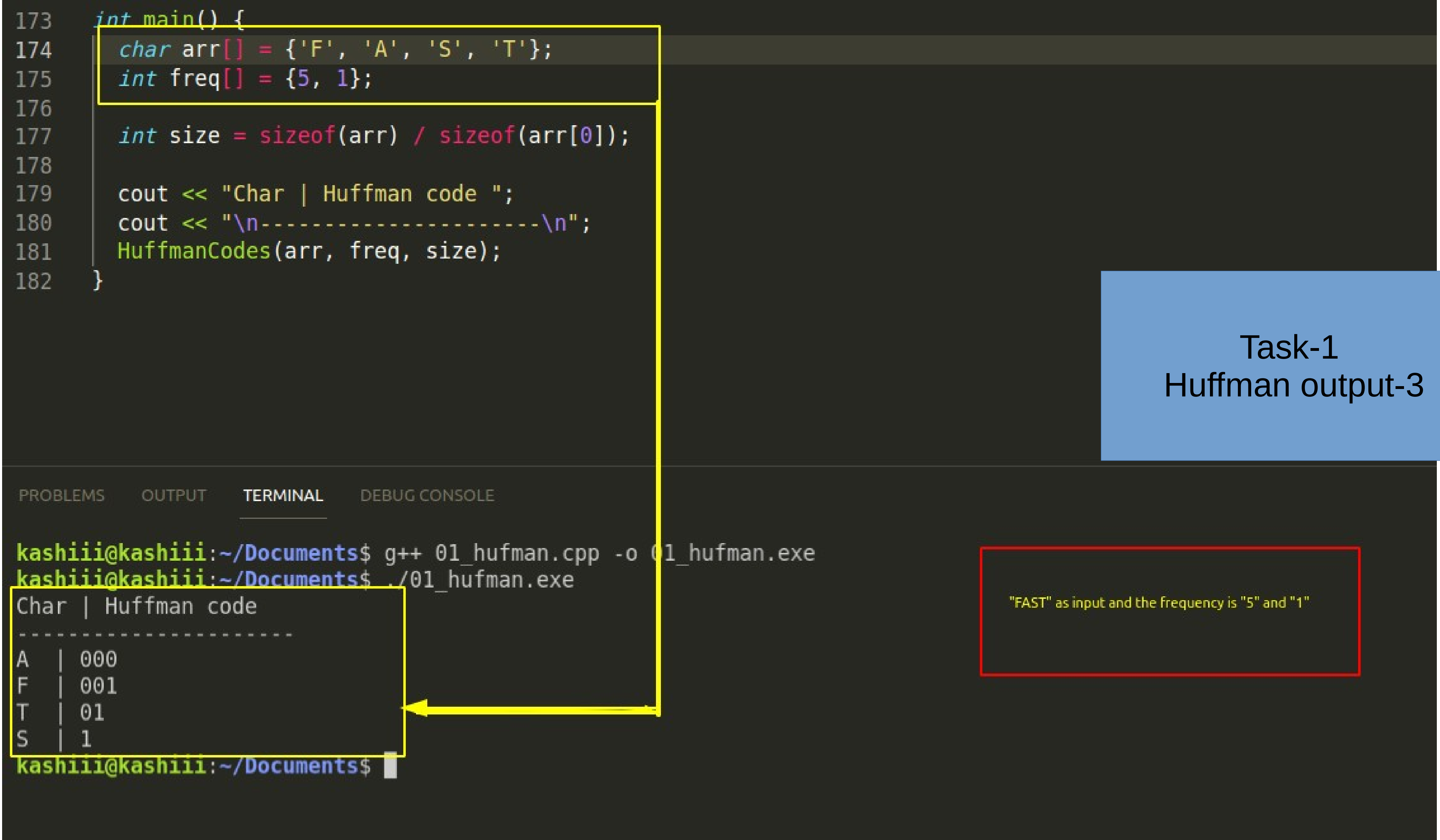
Task-1
Huffman output-1



```cpp
int main() {
    char arr[] = {'k', 'a', 's', 'h','i','f'};
    int freq[] = {5, 1, 6, 3};

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

    cout << "Char | Huffman code ";
    cout << "\n----------------------\n";
    HuffmanCodes(arr, freq, size);
}
```

```
kashiii@kashiii:~/Documents$ g++ 01_hufman.cpp -o 01_hufman.exe
kashiii@kashiii:~/Documents$ ./01_hufman.exe
Char | Huffman code
----------------------
s  | 000
a  | 00100
h  | 00101
k  | 0011
i  | 01
f  | 1
kashiii@kashiii:~/Documents$
```

Task-1
Huffman output-2



```
int main() {
    char arr[] = {'F', 'A', 'S', 'T'};
    int freq[] = {5, 1};

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

    cout << "Char | Huffman code ";
    cout << "\n----------------------\n";
    HuffmanCodes(arr, freq, size);
}
```

```
kashiii@kashiii:~/Documents$ g++ 01_hufman.cpp -o 01_hufman.exe
kashiii@kashiii:~/Documents$ ./01_hufman.exe
Char | Huffman code
----------------------
A  | 000
F  | 001
T  | 01
S  | 1
kashiii@kashiii:~/Documents$
```

Task-1
Huffman output-3

"FAST" as input and the frequency is "5" and "1"

Thank You

...