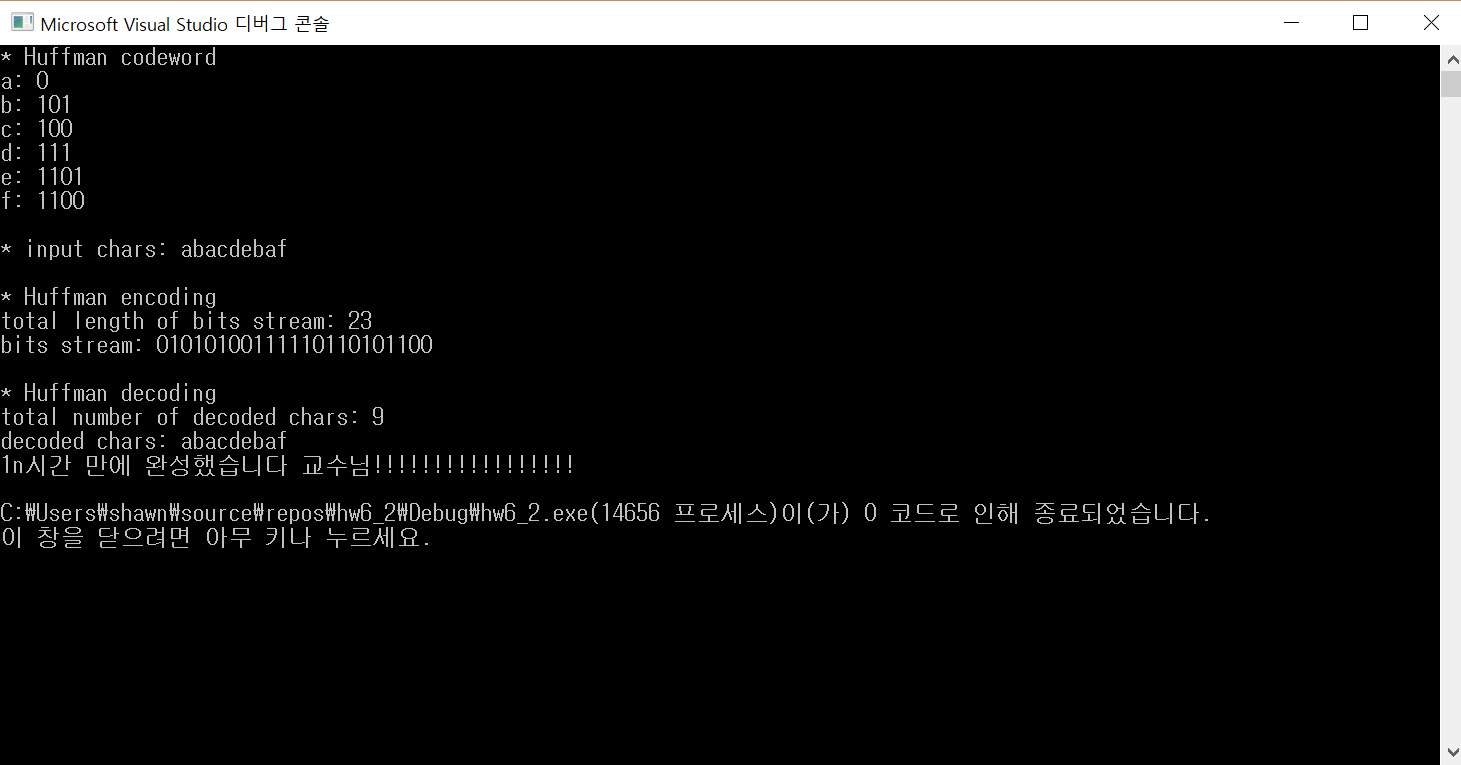
1. 코드 분석 및 주석
2. // huffman.cpp : Defines the entry point for the console application.
3. //
4. #include "stdlib.h"
5. #include "stdio.h"
6. #include "string.h"
7. #define MAX\_ELEMENT 1000
8. #define MAX\_BIT 10
9. #define MAX\_CHAR 20
10. // Input data for huffman code
11. typedef struct input\_huff {
12. char \*data; // Character array (a ~ f)
13. int \*freq; // Frequency array
14. int size; // Number of characters
15. } input\_huff;
16. // Structure for huffman binary tree
17. typedef struct TreeNode {
18. char data; // Character (a ~ f)
19. int key; // Frequency
20. int bits[MAX\_BIT]; // Huffman codeword
21. int bit\_size; // Huffman codeword's size
22. struct TreeNode \*l; // Left child of huffman binary tree
23. struct TreeNode \*r; // Right child of huffman binary tree
24. } TreeNode;
25. // Structure for bits stream
26. typedef struct bits\_stream {
27. int \*stream;
28. int length;
29. } bits\_stream;
30. // Elements used in the heap
31. typedef struct element {
32. TreeNode \*ptree;
33. int key; // frequency of each character
34. } element;
35. // Heap
36. typedef struct HeapType {
37. element heap[MAX\_ELEMENT];
38. int heap\_size;
39. } HeapType;
40. int \*\*m\_LUT, \*m\_bit\_size;
41. int m\_char\_size = 6;
42. // Initialization
43. void init(HeapType \*h)
44. {
45. h->heap\_size = 0;
46. }
47. //
48. int is\_empty(HeapType \*h)
49. {
50. if (h->heap\_size == 0)
51. return 1;
52. else
53. return 0;
54. }
55. void insert\_min\_heap(HeapType \*h, element item)
56. {
57. int i;
58. i = ++(h->heap\_size);
59. // compare it with the parent node in an order from the leaf to the root
60. while ((i != 1) && (item.key < h->heap[i / 2].key)) {
61. h->heap[i] = h->heap[i / 2];
62. i /= 2;
63. }
64. h->heap[i] = item; // Insert new node
65. }
66. element delete\_min\_heap(HeapType \*h)
67. {
68. int parent, child;
69. element item, temp;
70. item = h->heap[1];
71. temp = h->heap[(h->heap\_size)--];
72. parent = 1;
73. child = 2;
74. while (child <= h->heap\_size) {
75. if ((child < h->heap\_size) && (h->heap[child].key) > h->heap[child + 1].key)
76. child++;
77. if (temp.key <= h->heap[child].key) break;
78. h->heap[parent] = h->heap[child];
79. parent = child;
80. child \*= 2;
81. }
82. h->heap[parent] = temp;
83. return item;
84. }
85. // Node generation in binary tree
86. TreeNode \*make\_tree(TreeNode \*left, TreeNode \*right)
87. {
88. TreeNode \*node = (TreeNode \*)malloc(sizeof(TreeNode));
89. if (node == NULL) {
90. fprintf(stderr, "Memory allocation error\n");
91. exit(1);
92. }
93. node->l = left;
94. node->r = right;
95. return node;
96. }
97. // Binary tree removal
98. void destroy\_tree(TreeNode \*root)
99. {
100. if (root == NULL) return;
101. destroy\_tree(root->l);
102. destroy\_tree(root->r);
103. free(root);
104. }
105. // Huffman code generation
106. element huffman\_tree(input\_huff \*huff)
107. {
108. int i;
109. TreeNode \*node, \*x;
110. HeapType heap;
111. element e, e1, e2;
112. init(&heap);
113. int n = huff->size;
114. for (i = 0; i < n; i++) {
115. node = make\_tree(NULL, NULL);
116. e.ptree = node;
117. node->data = huff->data[i];
118. e.key = node->key = huff->freq[i];
119. memset(node->bits, 0, sizeof(int)\*MAX\_BIT);
120. node->bit\_size = 0;
121. insert\_min\_heap(&heap, e);
122. }
123. for (i = 1; i < n; i++) {
124. // Delete two nodes with minimum values
125. e1 = delete\_min\_heap(&heap);
126. e2 = delete\_min\_heap(&heap);
127. // Merge two nodes
128. x = make\_tree(e1.ptree, e2.ptree);
129. e.ptree = x;
130. x->data = NULL;
131. e.key = x->key = e1.key + e2.key;
132. memset(x->bits, 0, sizeof(int)\*MAX\_BIT);
133. x->bit\_size = 0;
134. insert\_min\_heap(&heap, e);
135. }
136. e = delete\_min\_heap(&heap); // Final Huffman binary tree
137. return e;
138. // destroy\_tree(e.ptree);
139. }
140. // Generate the huffman codeword from the huffman binary tree
141. // Hint: use the recursion for tree traversal
142. // input: root node
143. // output: m\_LUT, m\_bit\_size
144. int k, j = 0;
145. void huffman\_traversal(TreeNode \*node,int temp[],int top)//root node와 임시배열, top을 새로 세워주었다.
146. {
147. if (node->l) {//왼쪽 노드가 존재한다면
148. temp[top] = 0;//임시 배열에 0을 집어넣어 준다.
149. huffman\_traversal(node->l,temp,top+1);//다시 재귀함수를 불러준다. 이 때 top을 증가시킴.
151. }
152. if (node->r) {//오른쪽 노드가 존재한다면
153. temp[top] = 1;//임시배열에 1을 넣어준다.
154. huffman\_traversal(node->r,temp,top+1);//다시 재귀함수.
155. }
156. if((node->l==NULL)&(node->r==NULL)) {//만약 도착한 노드가 리프노드라면
157. switch (node->data) {//a=0,b=1 이렇게 지정해주고 난 다음
158. case 'a':
159. k = 0;
160. break;
161. case 'b':
162. k = 1;
163. break;
164. case 'c':
165. k = 2;
166. break;
167. case 'd':
168. k = 3;;
169. break;
170. case 'e':
171. k = 4;
172. break;
173. case 'f':
174. k = 5;
175. break;
176. }
177. m\_bit\_size[k] = top;
178. for (j = 0; j < top; j++) {//미리 선언한 2차원 배열에 코드를 넣어준다.
179. m\_LUT[k][j] = temp[j];
180. }
181. }
183. }
184. int \*\*mem\_2D\_int(int row, int col)
185. {
186. int \*\*m2 = (int \*\*)malloc(sizeof(int \*)\*row);
187. for (int i = 0; i < row; i++)
188. m2[i] = (int \*)malloc(sizeof(int)\*col);
189. return m2;
190. }
191. void print\_codeword()
192. {
193. printf("\* Huffman codeword\n");
194. for (int i = 0; i < m\_char\_size; i++)
195. {
196. switch (i) {
197. case 0:
198. printf("%c: ", 'a');
199. break;
200. case 1:
201. printf("%c: ", 'b');
202. break;
203. case 2:
204. printf("%c: ", 'c');
205. break;
206. case 3:
207. printf("%c: ", 'd');
208. break;
209. case 4:
210. printf("%c: ", 'e');
211. break;
212. case 5:
213. printf("%c: ", 'f');
214. break;
215. }
216. for (int j = 0; j < m\_bit\_size[i]; j++)
217. printf("%d", m\_LUT[i][j]);
218. printf("\n");
219. }
220. }
221. // Input: 'str'
222. // Output: 'bits\_stream' (consisting of 0 or 1)
223. // 'bits\_stream' is generated using 'm\_LUT' generated by the huffman binary tree
224. // Return the total length of bits\_stream
225. void huffman\_encoding(char \*str, bits\_stream \*bits\_str)
226. {
227. int interval = 0;
228. int j = 0;//스트링 알파벳 갯수
229. int k = 0;
230. int l = strlen(str);//스트링 길이
231. int arr[9];//알파벳 숫자를 담아두는 곳
232. for (j = 0; j < l; j++) {//각각 알파벳에 알맞은 2차원 행 입력
233. switch (str[j]) {
234. case'a':
235. arr[j] = 0;
236. break;
237. case'b':
238. arr[j] = 1;
239. break;
240. case'c':
241. arr[j] = 2;
242. break;
243. case'd':
244. arr[j] = 3;
245. break;
246. case'e':
247. arr[j] = 4;
248. break;
249. case'f':
250. arr[j] = 5;
251. break;
252. }
253. }
254. for(j=0;j<l;j++){//각각의 코드 bits\_Str에 넣어주기
255. while (k<m\_bit\_size[arr[j]]) {
256. bits\_str->stream[interval++] = m\_LUT[arr[j]][k++];
257. }
258. k = 0;
259. }
261. printf("\n\* Huffman encoding\n");
262. printf("total length of bits stream: %d\n", interval);
263. printf("bits stream: ");
264. for (int i = 0; i < interval; i++)
265. printf("%d", bits\_str->stream[i]);
266. printf("\n");
267. bits\_str->length = interval;
269. }
270. // input: 'bits\_stream' and 'total\_length'
271. // output: 'decoded\_str'
272. int huffman\_decoding(bits\_stream \*bits\_str, TreeNode \*node, char \*decoded\_str)
273. {
274. int i = 0;
275. int j = 0;
276. TreeNode \*temp = node;
277. int index\_char=0;
278. for (i = 0; i < bits\_str->length; i++) {//트리 따라가기
279. if (bits\_str->stream[i] == 0) {//만약 배열이 0이 있다면
280. temp = temp->l;//왼쪽을 따라간다.
281. if (temp->l == NULL && temp->r == NULL) {//만약 leaf 노드에 도착했다면
282. decoded\_str[j++] = temp->data;//문자를 저장해주고
283. index\_char++;//해독된 문자 count 증가.
284. temp = node;//노드는 다시 루트노드로 리셋.
285. }
286. }
287. else if (bits\_str->stream[i] == 1) {//만약 배열에 1이 있다면
288. temp = temp->r;//오른쪽을 따라간다.
289. if (temp->l == NULL && temp->r == NULL) {//똑같이 leat노드에 도착
290. decoded\_str[j++] = temp->data;
291. index\_char++;
292. temp = node;
293. }
294. }
295. }
296. printf("\n\* Huffman decoding\n");
297. printf("total number of decoded chars: %d\n", index\_char);
298. printf("decoded chars: ");
299. for (int i = 0; i < index\_char; i++)
300. printf("%c", decoded\_str[i]);
301. printf("\n");
302. return index\_char;
303. }
304. void main()
305. {
306. int arr[MAX\_BIT];
307. char data[] = { 'a', 'b', 'c', 'd', 'e', 'f' };
308. int freq[] = { 45, 13, 12, 16, 9, 5 };
309. input\_huff \*huff1 = (input\_huff \*)malloc(sizeof(input\_huff));
310. huff1->data = data;
311. huff1->freq = freq;
312. huff1->size = m\_char\_size;
313. // m\_LUT: each row corresponds to the codeword for each character
314. // m\_bit\_size: 1D array of codeword size for each character
315. // For instance, a = 0, b = 101, ...
316. // 1st row of 'm\_LUT': 0 0 ... 0
317. // 2nd row of 'm\_LUT': 1 0 1 ...0
318. // m\_bit\_size = {1, 3, ...}
319. m\_LUT = mem\_2D\_int(m\_char\_size, MAX\_BIT);
320. m\_bit\_size = (int \*)malloc(sizeof(int)\*m\_char\_size);
321. // Generate the huffman binary tree on heap
322. // 'element\_root': element containing the root node
323. element element\_root = huffman\_tree(huff1);
324. // Generate the huffman codeword from the huffman binary tree
325. huffman\_traversal(element\_root.ptree,arr,0);
326. //printf out the huffman codeword
327. print\_codeword();
328. //example of input data
329. char str[MAX\_CHAR] = { "abacdebaf" };
330. //char str[MAX\_CHAR] = { "ab" };
331. char decoded\_str[MAX\_CHAR];
332. printf("\n\* input chars: ");
333. for (int i = 0; i < strlen(str); i++)
334. printf("%c", str[i]);
335. printf("\n");
336. //start encoding
337. bits\_stream \*bits\_str1 = (bits\_stream \*)malloc(sizeof(bits\_stream));
338. bits\_str1->stream = (int \*)malloc(sizeof(int)\*MAX\_BIT \* MAX\_CHAR);
339. memset(bits\_str1->stream, -1, sizeof(int)\*MAX\_BIT \* MAX\_CHAR);
340. bits\_str1->length = 0;
341. huffman\_encoding(str, bits\_str1);
342. //start decoding
343. int decoded\_char\_length = huffman\_decoding(bits\_str1, element\_root.ptree, decoded\_str);
344. printf("1n시간 만에 완성했습니다 교수님!!!!!!!!!!!!!!!!!\n");
345. }

2. 결과 및 콘솔 창



3. 코드 분석

1) traversal 함수 – 우선 재귀함수를 통해 만들었다. 가장 고민했던 부분은 parent node의 bit를 어떻게 child node에게 전달시키느냐의 문제였다. 일단 트리를 따라가면서 왼쪽으로 가면 0을, 오른쪽으로 가면 1을 저장해주고 leaf node에 도착하면 임시배열을 그대로 저장하는 방법을 했다. 하지만 이건 재취함수가 아니었다면 할 수 없었을 것이다. B와 c는 마지막 0과 1차이 밖에 없는데 그 전의 비트는 똑같이 복사하고 마지막 노드만 어떻게 다르게 만드냐의 문제였다. 그래서 재귀함수를 통해 top으로 2차원 배열 열을 조정했다. 또한 swtich문에 맞게 각각 a=0,b=1,c=2 이렇게 입력해주었다.

1. void huffman\_traversal(TreeNode \*node,int temp[],int top)//root node와 임시배열, top을 새로 세워주었다.
2. {
3. if (node->l) {//왼쪽 노드가 존재한다면
4. temp[top] = 0;//임시 배열에 0을 집어넣어 준다.
5. huffman\_traversal(node->l,temp,top+1);//다시 재귀함수를 불러준다. 이 때 top을 증가시킴.
7. }
8. if (node->r) {//오른쪽 노드가 존재한다면
9. temp[top] = 1;//임시배열에 1을 넣어준다.
10. huffman\_traversal(node->r,temp,top+1);//다시 재귀함수.
11. }
12. if((node->l==NULL)&(node->r==NULL)) {//만약 도착한 노드가 리프노드라면
13. switch (node->data) {//a=0,b=1 이렇게 지정해주고 난 다음
14. case 'a':
15. k = 0;
16. break;
17. case 'b':
18. k = 1;
19. break;
20. case 'c':
21. k = 2;
22. break;
23. case 'd':
24. k = 3;;
25. break;
26. case 'e':
27. k = 4;
28. break;
29. case 'f':
30. k = 5;
31. break;
32. }
33. m\_bit\_size[k] = top;
34. for (j = 0; j < top; j++) {//미리 선언한 2차원 배열에 코드를 넣어준다.
35. m\_LUT[k][j] = temp[j];
36. }
37. }
39. }

2) encoding 함수-이것도 switch문을 통해 알파벳 마다 숫자를 지정하고 그에 따른 2차원 배열을 지정하여 encoding을 해주었다.

1. void huffman\_encoding(char \*str, bits\_stream \*bits\_str)
2. {
3. int interval = 0;
4. int j = 0;//스트링 알파벳 갯수
5. int k = 0;
6. int l = strlen(str);//스트링 길이
7. int arr[9];//알파벳 숫자를 담아두는 곳
8. for (j = 0; j < l; j++) {//각각 알파벳에 알맞은 2차원 행 입력
9. switch (str[j]) {
10. case'a':
11. arr[j] = 0;
12. break;
13. case'b':
14. arr[j] = 1;
15. break;
16. case'c':
17. arr[j] = 2;
18. break;
19. case'd':
20. arr[j] = 3;
21. break;
22. case'e':
23. arr[j] = 4;
24. break;
25. case'f':
26. arr[j] = 5;
27. break;
28. }
29. }
30. for(j=0;j<l;j++){//각각의 코드 bits\_Str에 넣어주기
31. while (k<m\_bit\_size[arr[j]]) {
32. bits\_str->stream[interval++] = m\_LUT[arr[j]][k++];
33. }
34. k = 0;
35. }
37. printf("\n\* Huffman encoding\n");
38. printf("total length of bits stream: %d\n", interval);
39. printf("bits stream: ");
40. for (int i = 0; i < interval; i++)
41. printf("%d", bits\_str->stream[i]);
42. printf("\n");
43. bits\_str->length = interval;
45. }

3) decoding 함수- decoding은 트리 따라가는 방법을 사용했다. 만약 0이 있다면 왼쪽을 따라가고 1이 있다면 오른쪽을 따라가도록 했다. 그렇게 따라가다가 leaf노드에 도착하면 그 안의 데이터를 저장하고 node를 다시 루트노드로 세팅하는 방법을 사용했다.

1. int huffman\_decoding(bits\_stream \*bits\_str, TreeNode \*node, char \*decoded\_str)
2. {
3. int i = 0;
4. int j = 0;
5. TreeNode \*temp = node;
6. int index\_char=0;
7. for (i = 0; i < bits\_str->length; i++) {//트리 따라가기
8. if (bits\_str->stream[i] == 0) {//만약 배열이 0이 있다면
9. temp = temp->l;//왼쪽을 따라간다.
10. if (temp->l == NULL && temp->r == NULL) {//만약 leaf 노드에 도착했다면
11. decoded\_str[j++] = temp->data;//문자를 저장해주고
12. index\_char++;//해독된 문자 count 증가.
13. temp = node;//노드는 다시 루트노드로 리셋.
14. }
15. }
16. else if (bits\_str->stream[i] == 1) {//만약 배열에 1이 있다면
17. temp = temp->r;//오른쪽을 따라간다.
18. if (temp->l == NULL && temp->r == NULL) {//똑같이 leat노드에 도착
19. decoded\_str[j++] = temp->data;
20. index\_char++;
21. temp = node;
22. }
23. }
24. }
25. printf("\n\* Huffman decoding\n");
26. printf("total number of decoded chars: %d\n", index\_char);
27. printf("decoded chars: ");
28. for (int i = 0; i < index\_char; i++)
29. printf("%c", decoded\_str[i]);
30. printf("\n");
31. return index\_char;
32. }