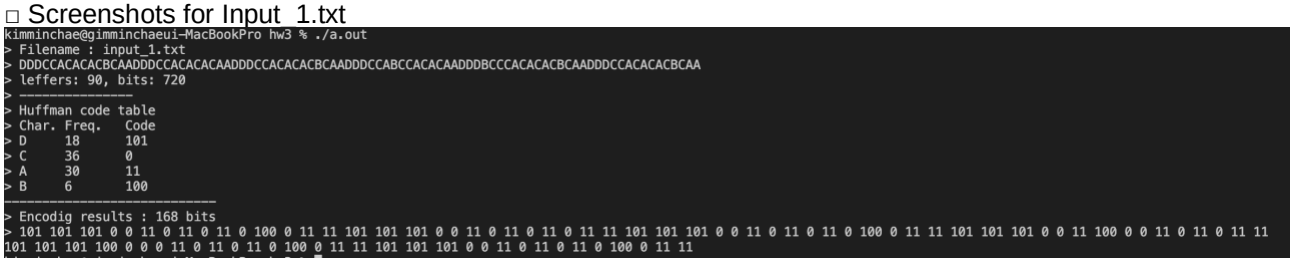
Algorithms analysis	Section	01
	Student number	22000080
Homework 3 – Huffuman code	Name	Kim, Min Chae

If your explanation is less informative and insufficient, then you may not get any points.
Also, you should provide discussion, otherwise you will get penalty.

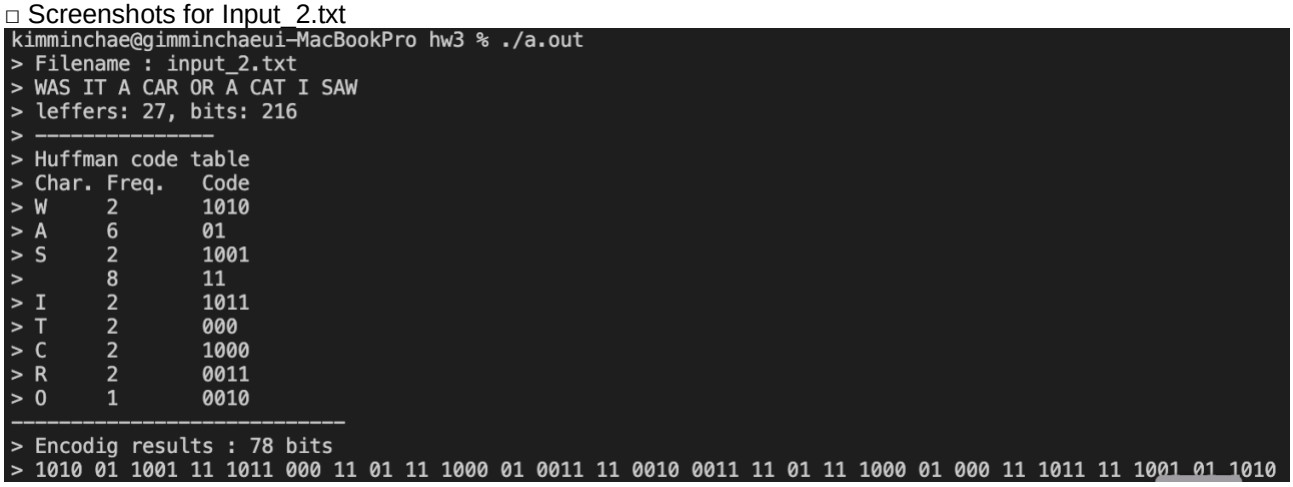
□ Information (complete the white cells)

Input file	Num. of letters	Num. of bits (simply multiply 8 to Num. of letters)	Num. of unique letters	Num. of Bits after Huffman code encoding
Input_sample.txt	15	120 bits	4	28 bits
Input_1.txt	90	720 bits	4	168 bits
Input_2.txt	27	216 bits	9	78 bits
Input_3.txt	319	2552 bits	32	1422 bits

□ Screenshots for Input_1.txt



□ Screenshots for Input_2.txt



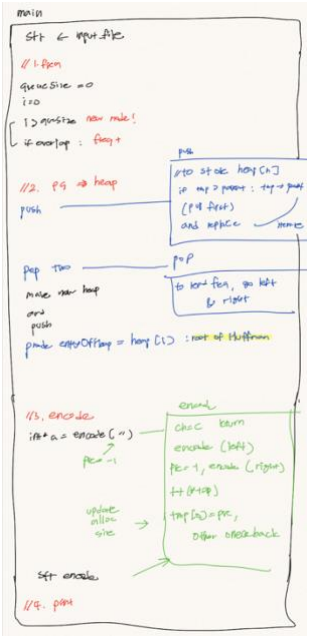
□ Screenshots for Input_3.txt

```
kimminchae@gimminchaeui-MacBookPro hw3 % ./a.out
> Filename : input_3.txt
> Moon river, wider than a mile
I'm crossing you in style some day
Oh, dream maker, you heart breaker
Wherever you're goin', I'm goin' your way
Two drifters, off to see the world
There's such a lot of world to see
We're after the same rainbow's end
Waitin' 'round the bend
My huckleberry friend
Moon river and me
> leffers: 319, bits: 2552
> ----------------
> Huffman code table
> Char. Freq.   Code
> M     3       0110111
> o     22      1001
> n     14      11111
>       48      110
> r     27      000
> i     13      11110
> v     3       1011000
> e     35      010
> ,     5       100010
> w     6       101110
> d     11      10101
> t     12      11100
> h     10      01111
> a     14      0010
> m     8       00110
> l     6       100011
>       9       00111
>
      9       01100
> I     2       10110111
> '     10      10000
> c     3       1110100
> s     11      10100
> g     3       1110101
> y     9       01110
> u     7       111011
> O     1       10110110
> k     3       1011001
> b     4       011010
> W     3       1011010
> T     2       0110110
> f     6       101111
----------------------------
> Encodig results : 1422 bits
> 0110111 1001 1001 11111 110 000 11110 1011000 010 000 100010 110 101110 11110 10101 010 000 110 11100 01111 0010 11111 110 0010 110 00110 11110 100011 010 00111 01100
 10110111 10000 00110 110 1110100 000 1001 10100 10100 11110 11111 1110101 110 01110 1001 111011 110 11110 11111 110 10100 11100 01110 100011 010 110 10100 1001 00110 0
10 110 10101 0010 01110 00111 01100 10110110 01111 100010 110 10101 000 010 0010 00110 110 00110 0010 1011001 010 000 100010 110 01110 1001 111011 110 01111 010 0010 00
0 11100 110 011010 000 010 0010 1011001 010 000 00111 01100 1011010 01111 010 000 010 1011000 010 000 110 01110 1001 111011 10000 000 010 110 1110101 1001 11110 11111 1
0000 100010 110 10110111 10000 00110 110 1110101 1001 11110 11111 10000 110 01110 1001 111011 000 110 101110 0010 01110 00111 01100 0110110 101110 1001 110 10101 000 11
110 101111 11100 010 000 10100 100010 110 1001 101111 101111 110 11100 1001 110 10100 010 010 110 11100 01111 010 110 101110 1001 000 100011 10101 00111 01100 0110110 0
1111 010 000 010 10000 10100 110 10100 111011 1110100 01111 110 0010 110 100011 1001 11100 110 1001 101111 110 101110 1001 000 100011 10101 110 11100 1001 110 10100 010
 010 00111 01100 1011010 010 10000 000 010 110 0010 101111 11100 010 000 110 11100 01111 010 110 10100 0010 00110 010 110 000 0010 11110 11111 011010 1001 101110 10000
10100 110 010 11111 10101 00111 01100 1011010 0010 11110 11100 11110 11111 10000 110 10000 000 1001 111011 11111 10101 110 11100 01111 010 110 011010 010 11111 10101 00
111 01100 0110111 01110 110 01111 111011 1110100 1011001 100011 010 011010 010 000 000 01110 110 101111 000 11110 010 11111 10101 00111 01100 0110111 1001 1001 11111 11
0 000 11110 1011000 010 000 110 0010 11111 10101 110 00110 010
```

□ Discussion about the results

- for input 1 and 2, it works well. But at 3, it shows two space except one '\n'. I think it is because of '\n' attack other nodes. But I think ' ' and '\n' is a one character, so it have to be shown. I tried to make node's char to string, and save there 'space' or 'newline', but it failed.

□ Discussion about the code



The left is the overall flow for the code. This first calculates the frequency. Second, it makes a heap. Third, it encodes. Finally, it prints out.

First, it calculates the frequency. Enter the file name from the user and read char one by one to fgetc. It then performs a frequency check on all characters in str. If i is greater than the initial qsize of 0, create a new node. This is a dynamic assignment, and it's used later to create a heap. If char already exists, increase freq.

Second, it makes a heap. Make a queue into a heap through Memcpy. At this point, the push and pop functions are used. When you push as much as Queuesize, check if the object you want to put in is larger than its parent, and if it is correct, make it a parent. You take the things in the back step by step and then you put them in. Then pull the two nodes out of the pop. This is where the pop function is used, and if the right side is small, it goes to the right. move to the nect lower level if freq of cur node smaller than tmp, goto left. And fill the hole made form pop.
It takes out two nodes like this and creates a new node that will be their parent. -1 to make a distinction between leaf and not leaf. Its frequency is sum of two node. The bigger is right. And push parent. entryIfHeap is huffman's root.

Thirdly, it encodes. The code function is called. If it's the data you want, return a null if it's a leaf but it's not what you want. The order is to visit the right first. If it is root, it returns what it receives. If there is no place to go, return null. If going left can't find it right. Afterwards, push back to insert new one to the first one.

Fourth, print out the results.

□ Codes // you should also submit the separate executable C or C++ files, TA will try run your code.

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>

#define MAX 1024

typedef struct node *pnode;
typedef struct node
{
  char ch;
  int freq;
  pnode left;
  pnode right;
}node;

void push( pnode* heap, const int n);
pnode pop( pnode* heap, const int n);
int* encode( pnode current, int pre, int* top, char c );

int main(){
        int i=0, qSize=0;
        char str[MAX];
        pnode pq[MAX];
        pnode heap[MAX];
        pnode entryOfHeap=NULL;
        FILE *fp;
        char filename[16];

        printf("> Filename : ");
        scanf("%s", filename);
```

```
if( ( fp=fopen(filename, "r+" ))==NULL ){
        printf("Input open error!\n" );
        exit(0);
}

char tmpc;
int stridx = 0;
while((tmpc = fgetc(fp)) != EOF){
        str[stridx++] = tmpc;
}
str[stridx] = '\0';
fclose(fp);

// calculate freq
for( int j=0; j<strlen(str); j++ ){
        // overlap
        for( i=1; i<=qSize && i<MAX ;i++ ){
                if( pq[i]->ch==str[j] ){
                        pq[i]->freq++;
                        break;
                }
        }
        //else
        if( i==qSize+1 )
        {
                pq[qSize+1]=(pnode)malloc( sizeof(node) );
                pq[qSize+1]->ch=str[j];
                pq[qSize+1]->freq=1;
                pq[qSize+1]->left=NULL;
                pq[qSize+1]->right=NULL;
                qSize++;
        }
}
printf("> %s\n", str);
printf("> leffers: %ld, bits: %ld\n", strlen(str), strlen(str)*8);
```

```
// priorityQueue -> heap
memcpy( &heap, &pq, sizeof(pq) );
for( i=1; i<=qSize && i<MAX ;i++ )
        push( heap, i );

for( i=qSize; i>1; i-- ){
        pnode tmp1 = pop( heap, i );
        pnode tmp2 = pop( heap, i-1 );

        heap[i-1] = (pnode)malloc( sizeof(node) );
        heap[i-1]->ch = -1;
        heap[i-1]->freq = tmp1->freq + tmp2->freq;

        if( tmp1->freq > tmp2->freq ){
                heap[i-1]->left=tmp2;
                heap[i-1]->right=tmp1;
        }
        else{
                heap[i-1]->left=tmp1;
                heap[i-1]->right=tmp2;
        }

        push( heap, i-1 );
}

entryOfHeap = heap[1];

printf("> ---------------\n");
printf("> Huffman code table\n");
printf("> Char.\tFreq.\tCode\n");

for( i=1; i<=qSize && i<MAX ;i++ ){
        printf("> %c\t%d\t", pq[i]->ch, pq[i]->freq );
        int top=0, *a=NULL;
```

```
            a = encode( entryOfHeap, -1, &top, pq[i]->ch );
                for( int j=0; j<top; j++ ){
                    printf("%d", a[j] );
                }
                printf("\n" );
        }

        int wordCount=0;
        char total[MAX] = "";
        int totalindex = 0;
        for( i=0; i<strlen(str); i++ ){
            int top=0, *a=NULL;
            a = encode( entryOfHeap, -1, &top, str[i] );
            for( int j=0; j<top; j++ )
                total[totalindex++] = a[j] + '0';
            total[totalindex++] = ' ';
            wordCount+=top;
            free(a);
        }

        // Print result
        printf("--------------------------\n");
        printf("> Encodig results : %d bits\n", wordCount);

        printf("> %s", total);
        printf("\n");

        return 0;
}

//heap function
void push( pnode* heap, const int n){
        int i=n;
        pnode tmp = heap[n];
        while ( (i>1) && ( tmp->freq < heap[i/2]->freq) ){
```

```
            heap[i] = heap[i/2];
            i /= 2;
        }
        heap[i] = tmp;
}

pnode pop( pnode* heap, const int n){
        int parent, child;
        pnode item, temp;
        item = heap[1];
        temp = heap[n];
        parent = 1;
        child = 2;
        while (child <= n){
            if ( (child < n) && ( heap[child]->freq > heap[child+1]->freq) )
                child++;
            if ( temp->freq <= heap[child]->freq )
                break;
            heap[parent] = heap[child];
            parent = child;
            child *= 2;
        }
        heap[parent] = temp;
        return item;
}

int* encode( pnode current, int pre, int* top, char c ){
        int* tmp;
        if( current->ch==c ){
            tmp=malloc( sizeof(int) );
            *tmp=pre;
            (*top)=1;
            return tmp;
        }
        if( current->ch!=-1 )
```

```
            return NULL;
        tmp=encode( current->left, 0, top, c );
        if( pre==-1 ){
            if( tmp!=NULL )
                return tmp;
            return encode( current->right, 1, top, c );
        }
        if( tmp==NULL ){
            tmp=encode( current->right, 1, top, c );
            if( tmp==NULL )
                return NULL;
        }
        ++(*top);
        tmp=realloc( tmp, *top );
        for( int i=(*top)-1; i>0; i-- )
            tmp[i]=tmp[i-1];
        tmp[0]=pre;
        return tmp;
}
```