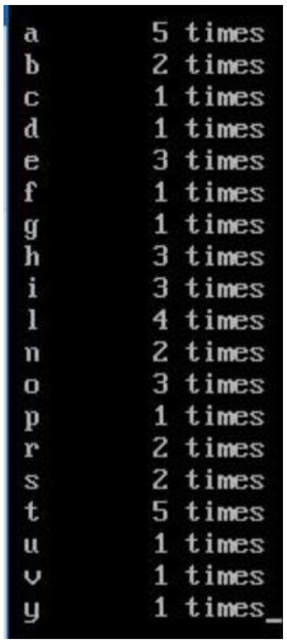
Practical 1

Write a program to get frequency count of all symbols in a file.

```
#include<stdio.h>
void main()
{	FILE *f;
	char c;
	int count[26]={0};
	int x, i;
	clrscr();
	f=fopen("f1.txt","r");
	while(c!=EOF)
	{	c=getc(f);
		x=c-97;
		count[x]++;
	}
	for(i=0;i<26;i++)
	{	if(count[i]!=0)
		printf("\n %c \t %d times",i+97,count[i]);
	}
	fclose(f);
	getch();
}
```

Output



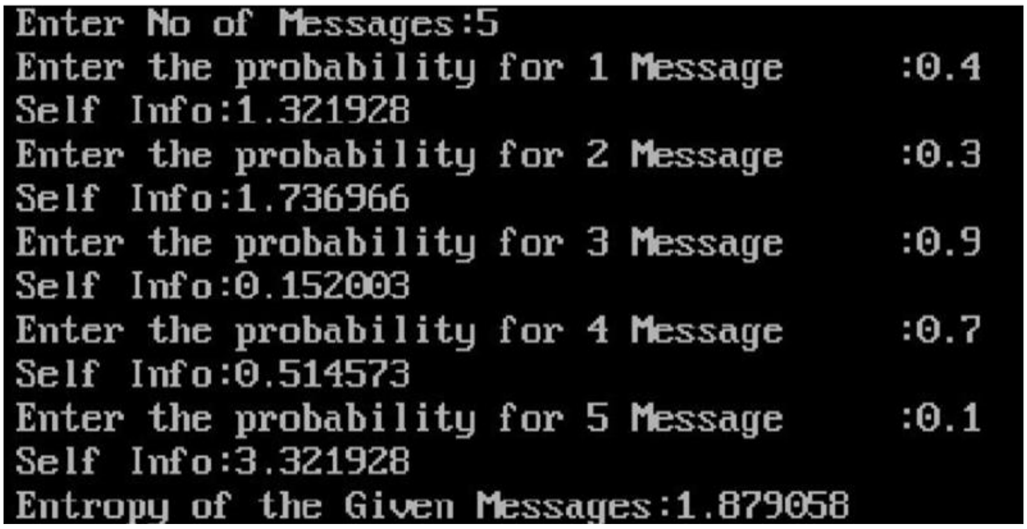
```
a         5 times
b         2 times
c         1 times
d         1 times
e         3 times
f         1 times
g         1 times
h         3 times
i         3 times
l         4 times
n         2 times
o         3 times
p         1 times
r         2 times
s         2 times
t         5 times
u         1 times
v         1 times
y         1 times_
```

Practical 2

Write a program to calculate self-information and entropy if probabilities are given.

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
void main()
{       int i, c;
        float p, sum=0,it;
        float selfin[50];
        clrscr();
        printf("Enter No of Messages:");
        scanf("%d",&c);
        for(i=1;i<=c;i++)
        {       printf("Enter the probability for %d Message\t:",i);
                scanf("%f",&p);
                selfin[i]=(log(1/p))/log(2);
                printf("Self Info:%f\n",selfin[i]);
                it=p*(log(1/p))/log(2);
                sum=sum+it;
        }
        printf("Entropy of the Given Messages:%f",sum);
        getch();
}
```

Output



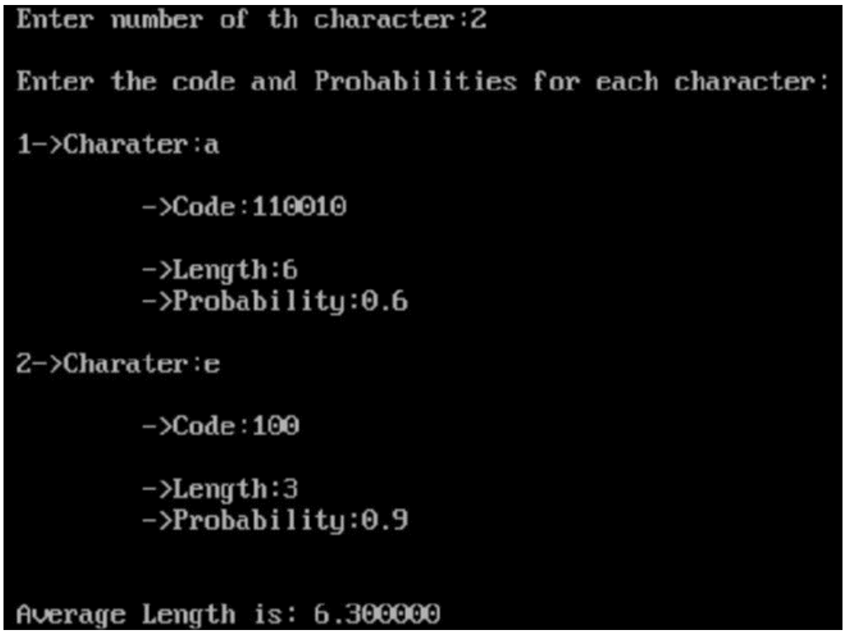
```
Enter No of Messages:5
Enter the probability for 1 Message      :0.4
Self Info:1.321928
Enter the probability for 2 Message      :0.3
Self Info:1.736966
Enter the probability for 3 Message      :0.9
Self Info:0.152003
Enter the probability for 4 Message      :0.7
Self Info:0.514573
Enter the probability for 5 Message      :0.1
Self Info:3.321928
Entropy of the Given Messages:1.879058
```

Practical 3

Write a program to calculate average length if probabilities and codes are given.

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
void main()
{       char str[50][50],c[50];
        float prob[50];
        int len[50],i=0,n=0;
        float avglen=0;
        clrscr();
        printf("Enter number of the character:");
        scanf("%d",&n);
        printf("\nEnter the code and Probabilities for each character:\n");
        for(i=0;i<n;i++)
        {       printf("\n%d->Charater:",i+1);
                scanf("%s",&c[i])
                printf("\n\t->Code:");
                scanf("%s",&str[i]);
                len[I]=strlen(str[i]);
                printf("\n\t->Length:%d",len[i]);
                printf("\n\t->Probability:");
                scanf("%f",&prob[i]);
                avglen=avglen+(prob[i]*len[i]);
        }
        printf("\n\nAverage Length is: %f",avglen);
        getch();
}
```

Output



```
Enter number of th character:2

Enter the code and Probabilities for each character:

1->Charater:a

        ->Code:110010

        ->Length:6
        ->Probability:0.6

2->Charater:e

        ->Code:100

        ->Length:3
        ->Probability:0.9


Average Length is: 6.300000
```

Practical 4

Implement Huffman Algorithm to generate code when symbol and probabilities are given.

```
#include<iostream.h>
#include<string.h>
#include<math.h>
#include<stdlib.h>
#include<conio.h>
struct tree
{       char a[20];
        int s;
        struct tree *left,*right;
}*root=NULL,*tt[20]={NULL},*temp,*temp2,*t2,*ri,*le;
struct pqu
{       int info;
        char a[20];
        struct pqu *ptr;
}*front=NULL,*t,*par,*t1,*p1,*p2;
struct pqu* fp(int info)
{       struct pqu *p=NULL;
        for(t1=front;t1->info<info&&t1!=NULL;t1=t1->ptr)
        {       p=t1;   }
        return (p);
}
void enqu(char a[20],int p)
{       t=(struct pqu*)malloc(sizeof(struct pqu));
        strcpy(t->a,a);
        t->info=p;
        t->ptr=NULL;
        if(front==NULL)
```

```
		{	front=t;}
		else
		{	par=fp(p);
			if(par==NULL)
			{	t->ptr=front;
				front=t;
			}
			else
			{	t->ptr=par->ptr;
				par->ptr=t;
			}
		}
}
struct pqu* dequ()
{	t1=front;
	front=front->ptr;
	return t1;
}
void info(char c[2])
{	int m=0,i;
	temp2=root;
	while(strcmp(c,temp2->a)!=0)
	{	t2=temp2->left;
		for(i=0;i<strlen(t2->a);i++)
		{	if(t2->a[i]==c[0])
			{	temp2=temp2->left;
				m=1;
				cout<<"0";
				break;
			}
		}
```

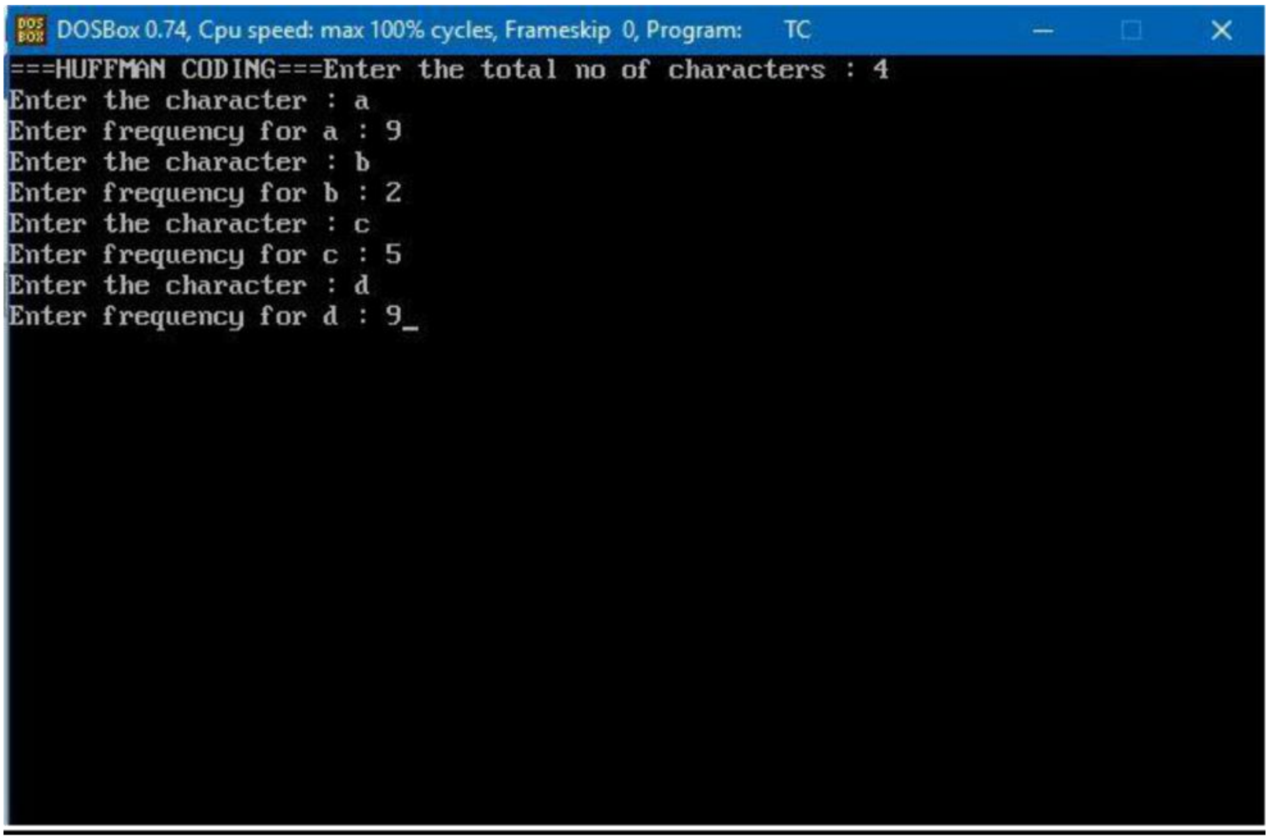
```
                        if(m!=1)
                        {       temp2=temp2->right;
                                cout<<1;
                        }
                        m=0;
                }
        }
        void insert()
        {       char a1[20],b1[20],v1[20];
                int i,j,z=0,l;
                while(front!=NULL)
                {       p1=dequ();
                        strcpy(a1,p1->a);
                        l=p1->info;
                        p2=dequ();
                        if(p2==NULL)
                                break;
                        strcpy(b1,p2->a);
                        strcpy(v1,a1);
                        temp=(struct tree*)malloc(sizeof(struct tree));
                        strcpy(temp->a,strcat(v1,b1));
                        temp->s=l+p2->info;
                        temp->left=NULL;
                        temp->right=NULL;
                        temp2=temp;
                        root=temp;
                        for(i=0;i<z;)
                        {       if(strcmp(tt[i]->a,a1)==0)
                                {       temp->left=tt[i];
                                        for(l=i;l<z;l++)
                                        {       tt[l]=tt[l+1];  }
```

```
				i=0;
				continue;
			}
			else if(strcmp(tt[i]->a,b1)==0)
			{	temp->right=tt[i];
				for(l=i;l<z;l++)
				{	tt[l]=tt[l+1];	}
				i=0;
				continue;
			}
			i++;
		}
		if(temp->left==NULL)
		{	le=(struct tree*)malloc(sizeof(struct tree));
			strcpy(le->a,a1);
			le->left=NULL;
			le->right=NULL;
			temp2->left=le;
		}
		if(temp->right==NULL)
		{	ri=(struct tree*)malloc(sizeof(struct tree));
			strcpy(ri->a,b1);
			ri->left=NULL;
			ri->right=NULL;
			temp2->right=ri;
		}
		if(front!=NULL)
		{	enqu(temp2->a,temp2->s);
			tt[z++]=temp2;
		}
	}
```

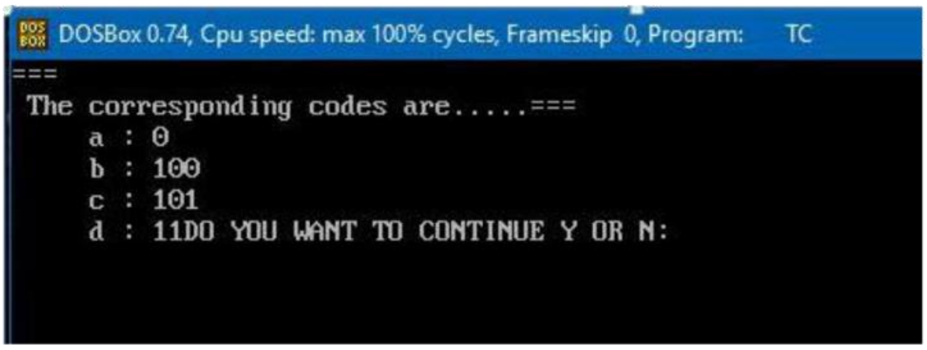
```
}
void disp(struct tree *rt)
{       if(rt!=NULL)
        {       disp(rt->left);
                cout<<" "<<rt->a;
                disp(rt->right);
        }
}
void main()
{       textmode(MONO);
        int i=0,g,h,p,y,n;
        char m[20],b[20][2],re;
        while(1)
        {       clrscr();
                cout<<"===";
                cout<<"\n HUFFMAN CODING";
                cout<<"===";
                cout<<"\n Enter the total no of characters : ";
                cin>>n;
                for(i=0;i<n;i++)
                {       cout<<"Enter the character : ";
                        cin>>m;
                        strcpy(b[i],m);
                        cout<<"Enter frequency for "<<m<<" : ";
                        cin>>g;
                        enqu(m,g);
                }
                insert();
                disp(root);
                clrscr();
                cout<<"===";
```

```
		cout<<"\n HUFFMAN CODING";
		cout<<"===";
		cout<<"\n Enter the total no of characters : ";
		cin>>n;
		for(i=0;i<n;i++)
		{	cout<<"Enter the character : ";
			cin>>m;
			strcpy(b[i],m);
			cout<<"Enter frequency for "<<m<<" : ";
			cin>>g;
			enqu(m,g);
		}
		insert();
		disp(root);
		clrscr();
		cout<<"===";
		cout<<"\n The corresponding codes are.....";
		cout<<"===";
		for(i=0;i<n;i++)
		{	cout<<"\n	"<<b[i]<<" : ";
			info(b[i]);
		}
		cout<<"DO YOU WANT TO CONTINUE Y OR N: ";
		cin>>re;
		if(re=='y'||re=='Y')
			continue;
		else
			exit(0);
	}
}
```

Output


DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC

```
===HUFFMAN CODING===Enter the total no of characters : 4
Enter the character : a
Enter frequency for a : 9
Enter the character : b
Enter frequency for b : 2
Enter the character : c
Enter frequency for c : 5
Enter the character : d
Enter frequency for d : 9_
```


DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC

```
===
 The corresponding codes are.....===
     a : 0
     b : 100
     c : 101
     d : 11DO YOU WANT TO CONTINUE Y OR N:
```

Practical 5

Implementation of Adaptive Huffman Algorithm

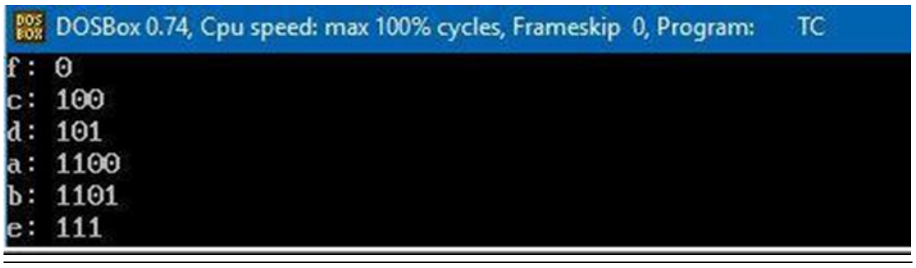
```
#include <stdio.h>

#include <stdlib.h>

#define MAX_TREE_HT 100

struct MinHeapNode

{	char data;

	unsigned freq;

	struct MinHeapNode *left, *right;

};

struct MinHeap

{	unsigned size;

	unsigned capacity;

	struct MinHeapNode **array;

};

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;

}

struct MinHeap* createMinHeap(unsigned capacity)

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

	minHeap->size = 0;

	// current size is 0 minHeap->capacity = capacity; minHeap->array =(struct
MinHeapNode**)malloc(minHeap->capacity * sizeof(struct MinHeapNode*)); return minHeap;

}

void swapMinHeapNode(struct MinHeapNode** a, struct MinHeapNode** b)

{	struct MinHeapNode* t = *a;

	*a = *b;
```

```
        *b = t;
}
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 funvtion 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);
	}
	void printArr(int arr[], int n)
	{	int i;
		for(i = 0; i < n; ++i)
			printf("%d", arr[i]);
		printf("\n");
	}
	int isLeaf(struct MinHeapNode* root)
	{	return !(root->left) && !(root->right) ;	}
	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;
	}
	struct MinHeapNode* buildHuffmanTree(char data[], int freq[], int size)
	{	struct MinHeapNode *left, *right, *top;
		struct MinHeap* minHeap = createAndBuildMinHeap(data, freq, size);
		while (!isSizeOne(minHeap))
		{	left = extractMin(minHeap);
			right = extractMin(minHeap);
			top = newNode('$', left->freq + right->freq);
```

```
                top->left = left;
                top->right = right;
                insertMinHeap(minHeap, top);
        }
        return extractMin(minHeap);
}
void printCodes(struct MinHeapNode* root, int arr[], int top)
{       if(root->left)
        {       arr[top] = 0;
                printCodes(root->left, arr, top + 1);
        }
        if(root->right)
        {       arr[top] = 1;
                printCodes(root->right, arr, top + 1);
        }
        if(isLeaf(root))
        {       printf("%c: ", root->data);
                printArr(arr, top);
        }
}
void HuffmanCodes(char data[], int freq[], int size)
{       struct MinHeapNode* root = buildHuffmanTree(data, freq, size);
        int arr[MAX_TREE_HT], top = 0; printCodes(root, arr, top);
}
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;
}
```

Output


DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC

```
f: 0
c: 100
d: 101
a: 1100
b: 1101
e: 111
```

Practical 6

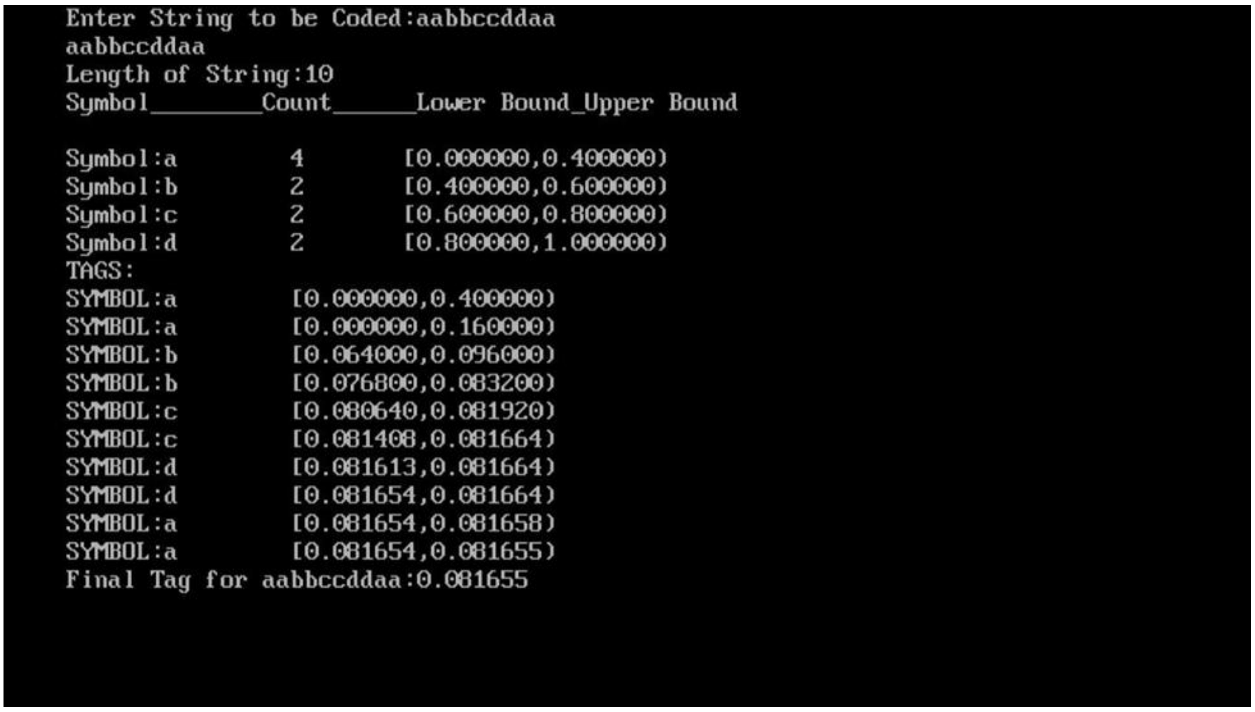
Implementation of Arithmetic Algorithm.

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
#include<math.h>
struct node
{       char c;
        int cnt;
        double p,lb,ub;
};
void ublb(int,double,double);
struct node n[100];
void main()
{       char ch,*input;
        double prob=0,lb,ub,UB=1,LB=0;
        int i,j,size=0,flag=0,t=0;
        clrscr();
        printf("Enter String to be Coded:");
        gets(input);
        while(size<strlen(input))
        {       printf("%c",input[size]);
                flag=0;
                for(j=0;j<t;j++)
                {       if(input[size]==n[j].c)
                        {       flag=1;
                                n[j].cnt++;
                                break;
                        }
                }
```

```
          if(flag==0)
          {         n[t].c=input[size];
                    n[t].cnt=1;
                    t++;
          }
          size++;
}
printf("\nLength of String:%d\n",size);
printf("Symbol________Count______Lower Bound_Upper Bound\n ");
prob=0;
for(i=0;i<t;i++)
{         n[i].p=n[i].cnt/(float)size;
          ublb(i,prob,n[i].p);
          prob+=n[i].p;
          printf("\nSymbol:%c\t%d\t[%f,%f)",n[i].c,n[i].cnt,n[i].lb,n[i].ub);
}
i=0;
printf("\nTAGS:");
while(i<strlen(input))
{         flag=0;
          ch=input[i];
          for(j=0;j<t;j++)
          {         if(ch==n[j].c)
                    {         lb=LB+(UB-LB)*(n[j].lb);
                              ub=LB+(UB-LB)*(n[j].ub);
                              break;
                    }
          }
          printf("\nSYMBOL:%c\t[%f,%f)",input[i],lb,ub);
          UB=ub;
          LB=lb;
```

```
                    i++;
          }
          printf("\nFinal Tag for %s:%f",input,(UB+LB)/2);
          getch();
}
void ublb(int i,double cp,double np)
{         if(cp<1)
          {         n[i].lb=cp;
                    n[i].ub=cp+np;
          }
}
```

Output



```
Enter String to be Coded:aabbccddaa
aabbccddaa
Length of String:10
Symbol________Count______Lower Bound_Upper Bound

Symbol:a        4       [0.000000,0.400000)
Symbol:b        2       [0.400000,0.600000)
Symbol:c        2       [0.600000,0.800000)
Symbol:d        2       [0.800000,1.000000)
TAGS:
SYMBOL:a        [0.000000,0.400000)
SYMBOL:a        [0.000000,0.160000)
SYMBOL:b        [0.064000,0.096000)
SYMBOL:b        [0.076800,0.083200)
SYMBOL:c        [0.080640,0.081920)
SYMBOL:c        [0.081408,0.081664)
SYMBOL:d        [0.081613,0.081664)
SYMBOL:d        [0.081654,0.081664)
SYMBOL:a        [0.081654,0.081658)
SYMBOL:a        [0.081654,0.081655)
Final Tag for aabbccddaa:0.081655
```

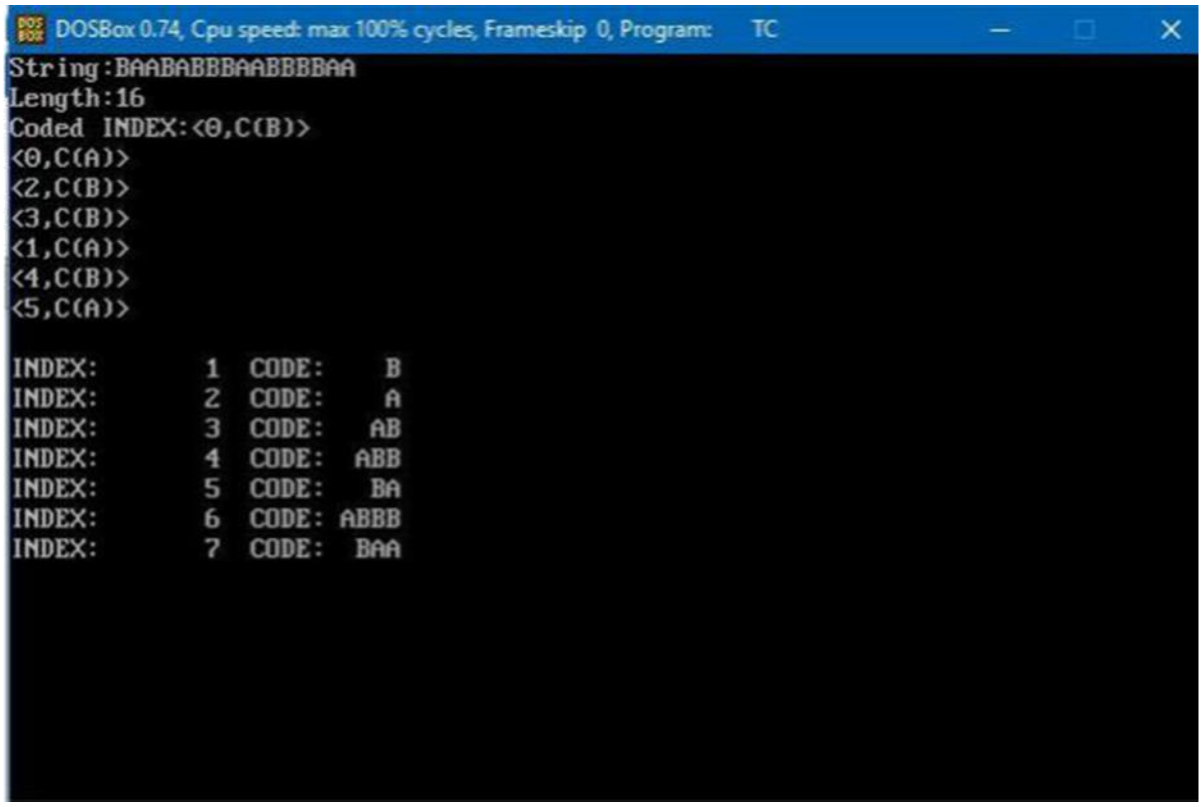
Practical 7

Implementation of lZ77 Algorithm

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
#include<stdlib.h>
char code[100][10];
char string[]="BAABABBBAABBBBAA";
int index=1;
void secondpass();
int check(char[],int);
void findindex(char[],int);
void printDict();
void main()
{           clrscr();
            printf("String:%s\n",string);
            printf("Length:%d\nCoded INDEX:",strlen(string));
            //finding individual symbols
            secondpass();
            printDict();
            getch();
}
void secondpass()
{           int i=0,j,flag=0,k;
            char tstring[10],tstr;
            while(string[i]!='\0')
            {           flag=0;
                        k=0;
                        tstring[0]=string[i];
                        while(!flag)
```

```
                {       tstr=tstring[k];
                        flag=check(tstring,k);
                        if(flag==1)
                        {       findindex(tstring,k);
                                printf("C(%c)>\n",tstr);
                                strncpy(code[index],tstring,k+1);
                                code[index][k+1]='\0';
                                index++;
                                i=i+k+1;
                                break;
                        }
                        else
                        {       k++;
                                if(string[i+k]!='\0')
                                {       tstring[k]=string[i+k];     }
                                else
                                {       findindex(tstring,k);
                                        i++;break;
                                }
                        }
                }
        }
}
int check(char *str,int k)
{       int i=1,j=0,flag;
        while(i<=index)
        {       flag=0;
        for(j=0;j<=k;j++)
        {       if (code[i][j]!=str[j])
                {       flag=1;
                        break;
                }
        }
        if(!flag) return 0;
                i++;
        return 1;
}
//OK
void printDict()
{       int i;
        for(i=1;i<index;i++)
        {       printf("\nINDEX:%8d\tCODE:%5s ",i,code[i]);     }
}
//OK
void findindex(char *str,int k)
{       int i=0,flag;
        while(i<=index)
        {       flag=0;
                flag=strncmp(code[i],str,k);
                if(flag==0)
                {       printf("<%d,",i);
                        return ;
                }
                else
                        i++;
        }
        return ;
}
```

Output


DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC

```
String:BAABABBBAABBBBAA
Length:16
Coded INDEX:<0,C(B)>
<0,C(A)>
<2,C(B)>
<3,C(B)>
<1,C(A)>
<4,C(B)>
<5,C(A)>

INDEX:       1  CODE:    B
INDEX:       2  CODE:    A
INDEX:       3  CODE:   AB
INDEX:       4  CODE:  ABB
INDEX:       5  CODE:   BA
INDEX:       6  CODE: ABBB
INDEX:       7  CODE:  BAA
```

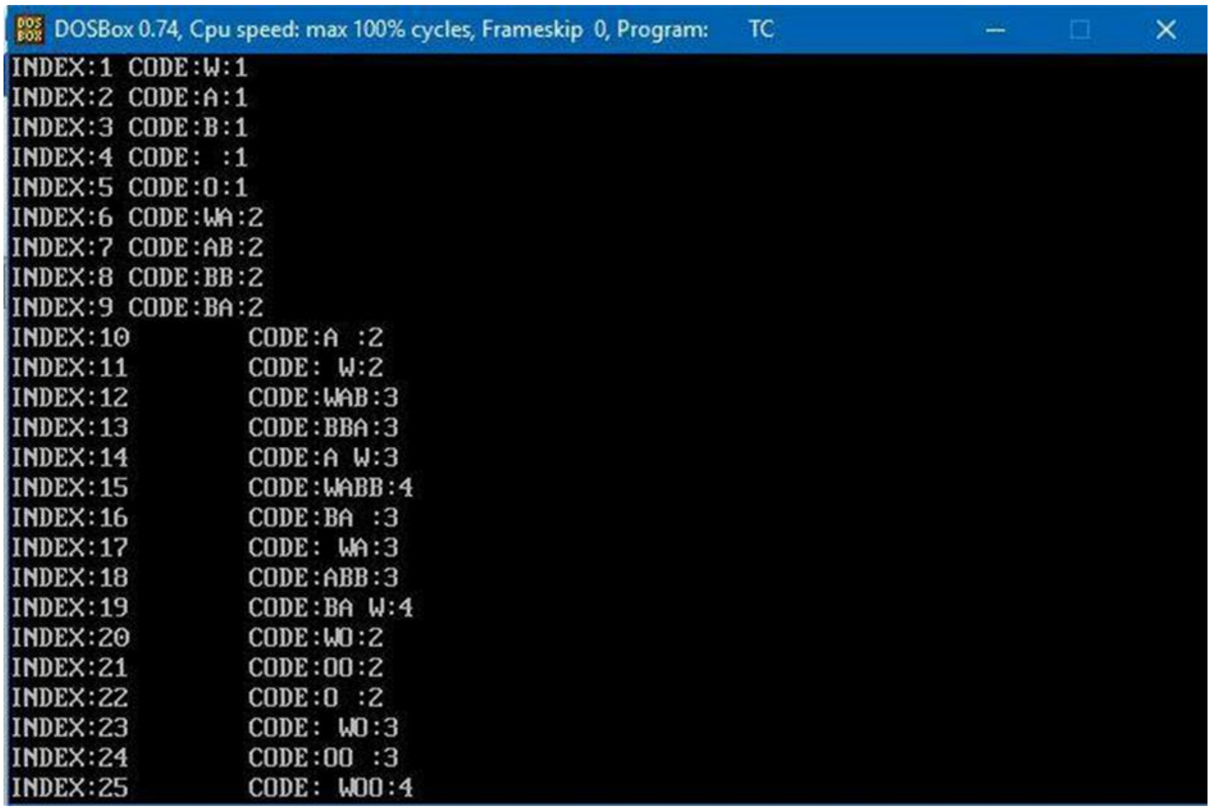
Practical 8

Implementation of LZ78 Algorithm.

```
#include<stdio.h>

#include<conio.h>

#include<string.h>

#include<stdlib.h>

char code[100][10];

char string[]="WABBA WABBA WABBA WABBA WOO WOO WOO";

int index=1;

void firstpass();

void secondpass();

int check(char[],int);

void findindex(char[],int);

void printDict();

void main()

{       clrscr();

        printf("String:%s\n",string);

        printf("Length:%d",strlen(string));

        //finding individual symbols

        printf("CALLING FIRST PASS\n");

        firstpass();

        printDict();

        printf("AFTER CALLING FIRST PASS\n");

        secondpass();

        printDict();

        getch();

}

//OK

void firstpass()

{       int i=0,j,flag;
```

```
        char ch;
        while(string[i]!='\0')
        {       ch=string[i];
                if(index==1)
                {       code[index][0]=ch;
                        index++;
                }
                else
                {       flag=1;
                        for(j=1;j<index;j++)
                        {       if(code[j][0]==ch)
                                {       flag=0;
                                        break;
                                }
                        }
                        if(flag)
                        {       code[index][0]=ch;
                                index++;
                        }
                }
                i++;
        }
}
void secondpass()
{       int i=0,j,flag=0,k;
        char tstring[10],tstr[10];
        while(string[i]!='\0')
        {       flag=0;
                k=0;
                tstring[0]=string[i];
                while(!flag)
                {       strncpy(tstr,tstring,k+1);
                        flag=check(tstring,k);
                        if(flag==1)
                        {       findindex(tstr,k);
                                strncpy(code[index],tstring,k+1);
                                code[index][k+1]='\0';
                                index++;
                                i=i+k
                                break;
                        }
                        else
                        {       k++;
                                if(string[i+k]!='\0')
                                {       tstring[k]=string[i+k];   }
                                else
                                {       findindex(tstr,k);
                                        i++;    break;
                                }
                        }
                }
        }
}
int check(char *str,int k)
{       int i=1,j=0,flag;
        while(i<index)
        {       flag=0;
                for(j=0;j<=k;j++)
                {       if(code[i][j]!=str[j])
                        {       flag=1;
                                break;
                        }
                }
                if(!flag) return 0;
                        i++;
        }
        return 1;
}
//OK
void printDict()
{       int i;
        for(i=1;i<index;i++)
        {       printf("\nINDEX:%d\tCODE:%s:%d",i,code[i],strlen(code[i]));     }
}
//OK
void findindex(char str[10],int k)
{       int i=1,flag;
        while(i<=index)
        {       flag=0;
                flag=strncmp(code[i],str,k);
                if(flag==0)
                {       printf("%d ",i);
                        return ;
                }
                else
                i++;
        }
        return ;
}
```

Output


DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC

```
INDEX:1 CODE:W:1
INDEX:2 CODE:A:1
INDEX:3 CODE:B:1
INDEX:4 CODE: :1
INDEX:5 CODE:O:1
INDEX:6 CODE:WA:2
INDEX:7 CODE:AB:2
INDEX:8 CODE:BB:2
INDEX:9 CODE:BA:2
INDEX:10        CODE:A :2
INDEX:11        CODE: W:2
INDEX:12        CODE:WAB:3
INDEX:13        CODE:BBA:3
INDEX:14        CODE:A W:3
INDEX:15        CODE:WABB:4
INDEX:16        CODE:BA :3
INDEX:17        CODE: WA:3
INDEX:18        CODE:ABB:3
INDEX:19        CODE:BA W:4
INDEX:20        CODE:WO:2
INDEX:21        CODE:OO:2
INDEX:22        CODE:O :2
INDEX:23        CODE: WO:3
INDEX:24        CODE:OO :3
INDEX:25        CODE: WOO:4
```

PRACTICAL 9

Implementation of JPEG Algorithm.

```
/* example.c

This file illustrates how to use the IJG code as a subroutine library to read or write JPEG image files.
You should look at this code in conjunction with the documentation file libjpeg.doc.

*

This code will not do anything useful as-is, but it may be helpful as a skeleton for constructing
routines that call the JPEG library.

*

We present these routines in the same coding style used in the JPEG code(ANSI function definitions,
etc.); but you are of course free to code your routines in a different style if you prefer. */


#include <stdio.h>


/* Include file for users of JPEG library.

You will need to have included system headers that define at least thetypedefs FILE and size_t
before you can include jpeglib.h.(stdio.h is sufficient on ANSI-conforming systems.)

You may also wish to include "jerror.h". */


#include "jpeglib.h"


/* <setjmp.h> is used for the optional error recovery mechanism shown in the second part of the
example. */


#include <setjmp.h>

/******************* JPEG COMPRESSION SAMPLE INTERFACE ******************/


/* This half of the example shows how to feed data into the JPEG compressor. We present a minimal
version that does not worry about refinements such as error recovery (the JPEG code will just exit() if
it gets an error). */

/* IMAGE DATA FORMATS:

The standard input image format is a rectangular array of pixels, with each pixel having the same
number of "component" values (color channels).Each pixel row is an array of JSAMPLEs (which
```

typically are unsigned chars).If you are working with color data, then the color values for each pixel must be adjacent in the row; for example, R,G,B,R,G,B,R,G,B,... for 24-bit RGB color.

* For this example, we'll assume that this data structure matches the way our application has stored the image in memory, so we can just pass a pointer to our image buffer. In particular, let's say that the image is RGB color and is described by: */

extern JSAMPLE *image_buffer;

/* Points to large array of R,G,B-order data */

externintimage_height;

/* Number of rows in image */

externintimage_width;

/* Number of columns in image */

/* Sample routine for JPEG compression. We assume that the target file name and a compression quality factor are passed in. */

GLOBAL(void)

write_JPEG_file (char * filename, int quality)

{

/* This struct contains the JPEG compression parameters and pointers to working space (which is allocated as needed by the JPEG library). It is possible to have several such structures, representing multiple compression/decompression processes, in existence at once. We refer to any one struct (and its associated working data) as a "JPEG object". */

structjpeg_compress_structcinfo;

/* This struct represents a JPEG error handler. It is declared separately because applications often want to supply a specialized error handler (see the second half of this file for an example). But here we just take the easy way out and use the standard error handler, which will print a message on

```
stderr and call exit() if compression fails. Note that this struct must live as long as the main JPEG
parameter struct, to avoid dangling-pointer problems. */

structjpeg_error_mgrjerr;

/* More stuff */

FILE * outfile;

/* target file */

JSAMPROW row_pointer[1];

/* pointer to JSAMPLE row[s] */

introw_stride;

/* physical row width in image buffer */
/* Step 1: allocate and initialize JPEG compression object */
/* We have to set up the error handler first, in case the initialization
step fails. (Unlikely, but it could happen if you are out of memory.)
This routine fills in the contents of structjerr, and returns jerr's
address which we place into the link field in cinfo.*/

cinfo.err = jpeg_std_error(&jerr);

/* Now we can initialize the JPEG compression object. */

jpeg_create_compress(&cinfo);

/* Step 2: specify data destination (eg, a file) */
/* Note: steps 2 and 3 can be done in either order. */

/* Here we use the library-supplied code to send compressed data to a stdio stream. You can also write your own code to do something else. VERY IMPORTANT: use "b" option to fopen() if you are on a machine that requires it in order to write binary files. */


if((outfile = fopen(filename, "wb")) == NULL)

{           fprintf(stderr, "can't open %s\n", filename);

            exit(1);

}

jpeg_stdio_dest(&cinfo, outfile);


/* Step 3: set parameters for compression */

/* First we supply a description of the input image.

Four fields of the cinfostruct must be filled in: */


cinfo.image_width = image_width;


/* image width and height, in pixels */


cinfo.image_height = image_height;

cinfo.input_components = 3;


/* # of color components per pixel */


cinfo.in_color_space = JCS_RGB;


/* colorspace of input image */

/* Now use the library's routine to set default compression parameters. (You must set at least cinfo.in_color_space before calling this, since the defaults depend on the source color space.)*/


jpeg_set_defaults(&cinfo);
```

```
/* Now you can set any non-default parameters you wish to. Here we just illustrate the use of
quality (quantization table) scaling: */

jpeg_set_quality(&cinfo, quality, TRUE /* limit to baseline-JPEG values */);

/* Step 4: Start compressor */

/* TRUE ensures that we will write a complete interchange-JPEG file. Pass TRUE unless you are very
sure of what you're doing. */


jpeg_start_compress(&cinfo, TRUE);


/* Step 5: while (scan lines remain to be written) */


        jpeg_write_scanlines(...);


/* Here we use the library's state variable cinfo.next_scanline as the loop counter, so that we don't
have to keep track ourselves. To keep things simple, we pass one scanline per call; you can pass
more if you wish, though.*/


row_stride = image_width * 3;


/* JSAMPLEs per row in image_buffer */


while (cinfo.next_scanline<cinfo.image_height)

{


/* jpeg_write_scanlines expects an array of pointers to scanlines. Here the array is only one element
long, but you could pass more than one scanline at a time if that's more convenient. */


        row_pointer[0] = &image_buffer[cinfo.next_scanline * row_stride];

        (void) jpeg_write_scanlines(&cinfo, row_pointer, 1);

}
```

```
/* Step 6: Finish compression */

jpeg_finish_compress(&cinfo);

/* Afterfinish_compress, we can close the output file. */

        fclose(outfile);

/* Step 7: release JPEG compression object */
/* This is an important step since it will release a good deal of memory. */

jpeg_destroy_compress(&cinfo);

/* And we're done! */

}

/* SOME FINE POINTS:
In the above loop, we ignored the return value of jpeg_write_scanlines, which is the number of
scanlines actually written. We could get away with this because we were only relying on the value of
cinfo.next_scanline,which will be incremented correctly. If you maintain additional loop variables
then you should be careful to increment them properly.Actually, for output to a stdio stream you
needn't worry, because thenjpeg_write_scanlines will write all the lines passed (or else exit with a
fatal error). Partial writes can only occur if you use a data destination module that can demand
suspension of the compressor.
(If you don't know what that's for, you don't need it.)
*
If the compressor requires full-image buffers (for entropy-coding optimization or a multi-scan JPEG
file), it will create temporary files for anything that doesn't fit within the maximum-memory setting.
(Note that temp files are NOT needed if you use the default parameters.) On some systems you may
need to set up a signal handler to ensure that temporary files are deleted if the program is
interrupted. See libjpeg.doc.Scanlines MUST be supplied in top-to-bottom order if you want your
JPEG files to be compatible with everyone else's. If you cannot readily read your data in that order,
you'll need an intermediate array to hold the image. See rdtarga.c or rdbmp.c for examples of
handling bottom-to-top source data using the JPEG code's internal virtual-array mechanisms. */
```

/******************* JPEG DECOMPRESSION SAMPLE INTERFACE *******************/

/* This half of the example shows how to read data from the JPEG decompressor.

It's a bit more refined than the above, in that we show:

(a) how to modify the JPEG library's standard error-reporting behavior;

(b) how to allocate workspace using the library's memory manager.

*

Just to make this example a little different from the first one, we'll assume that we do not intend to put the whole image into an in-memory buffer, but to send it line-by-line someplace else. We need a one-scanline-high JSAMPLE array as a work buffer, and we will let the JPEG memory manager allocate it for us. This approach is actually quite usefu because we don't need to remember to deallocate the buffer separately: it will go away automatically when the JPEG object is cleaned up. */

/* ERROR HANDLING:

The JPEG library's standard error handler (jerror.c) is divided into several "methods" which you can override individually. This lets you adjust the behavior without duplicating a lot of code, which you might have to update with each future release.

*

Our example here shows how to override the "error_exit" method so that control is returned to the library's caller when a fatal error occurs, rather than calling exit() as the standard error_exit method does.

*

We use C's setjmp/longjmp facility to return control. This means that the routine which calls the JPEG library must first execute a setjmp() call to establish the return point. We want the replacement error_exit to do a longjmp(). But we need to make the setjmp buffer accessible to the error_exit routine. To do this, we make a private extension of the standard JPEG error handler object. (If we were using C++, we'd say we're making a subclass of the regular error handler.)

*

Here's the extended error handler struct: */

```
structmy_error_mgr
{           structjpeg_error_mgr pub;


/* "public" fields */
```

```
        jmp_bufsetjmp_buffer;

/* for return to caller */

};
typedefstructmy_error_mgr * my_error_ptr;

/* Here's the routine that will replace the standard error_exit method: */

METHODDEF(void)
my_error_exit(j_common_ptrcinfo)
{

/*cinfo->err really points to a my_error_mgrstruct, so coerce pointer */

        my_error_ptrmyerr = (my_error_ptr) cinfo->err;

/* Always display the message. */
/* We could postpone this until after returning, if we chose. */

        (*cinfo->err->output_message) (cinfo);

/* Return control to the setjmp point */

        longjmp(myerr->setjmp_buffer, 1);
}

/* Sample routine for JPEG decompression. We assume that the source file name is passed in. We
want to return 1 on success, 0 on error.*/

GLOBAL(int)
```

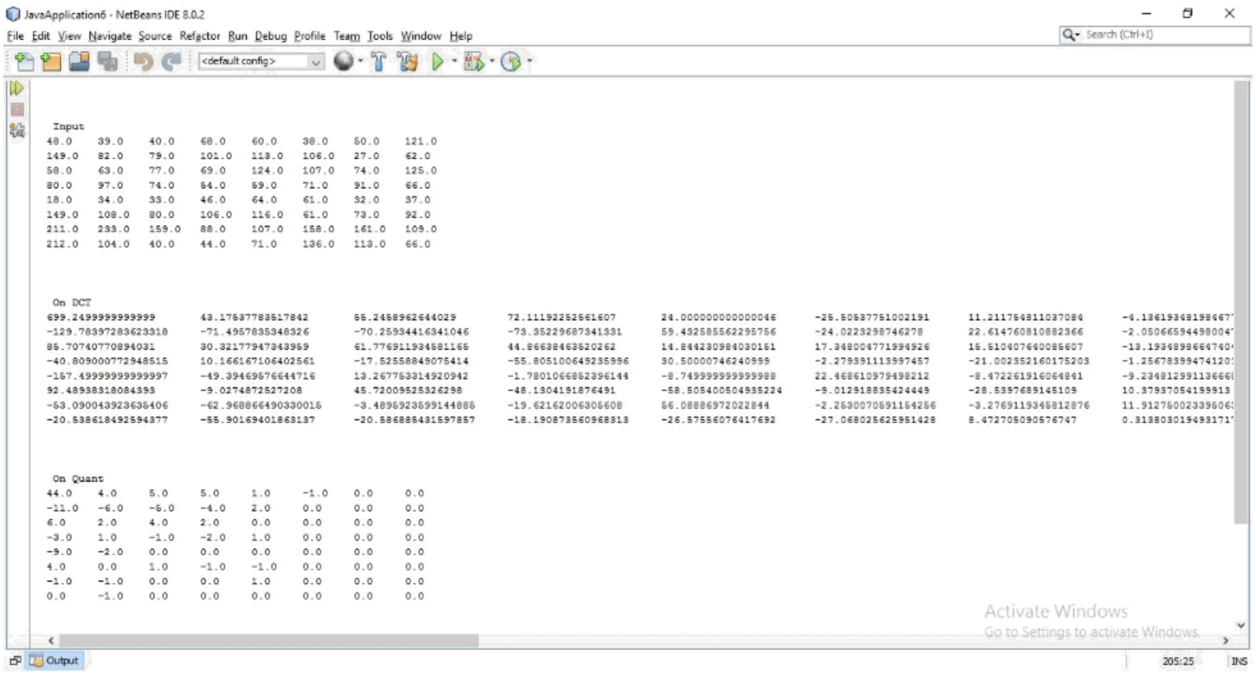
```
read_JPEG_file (char * filename)

{


/* Thisstruct contains the JPEG decompression parameters and pointers to

working space (which is allocated as needed by the JPEG library). */


            structjpeg_decompress_structcinfo;


/* We use our private extension JPEG error handler. Note that this struct must live as long as the
main JPEG parameter struct, to avoid dangling-pointer problems. */


            structmy_error_mgrjerr;


/* More stuff */


            FILE * infile;


/* source file */


            JSAMPARRAY buffer;


/* Output row buffer */


            introw_stride;


/* physical row width in output buffer */

/* In this example we want to open the input file before doing anything else, * so that the setjmp()
error recovery below can assume the file is open.

VERY IMPORTANT: use "b" option to fopen() if you are on a machine that requires it in order to read
binary files. */


            if((infile = fopen(filename, "rb")) == NULL)
```

```
        {           fprintf(stderr, "can't open %s\n", filename);
        return 0;
        }

/* Step 1: allocate and initialize JPEG decompression object
 We set up the normal JPEG error routines, then override error_exit. */

        cinfo.err = jpeg_std_error(&jerr.pub); jerr.pub.error_exit = my_error_exit;

/* Establish the setjmp return context for my_error_exit to use. */

        if (setjmp(jerr.setjmp_buffer))
        {

/* If we get here, the JPEG code has signaled an error.
We need to clean up the JPEG object, close the input file, and return. */

        jpeg_destroy_decompress(&cinfo);
        fclose(infile);
        return 0;
}

/* Now we can initialize the JPEG decompression object. */

jpeg_create_decompress(&cinfo);

/* Step 2: specify data source (eg, a file) */

jpeg_stdio_src(&cinfo, infile);

/* Step 3: read file parameters with jpeg_read_header() */
```

```
(void)jpeg_read_header(&cinfo, TRUE);

/* We can ignore the return value from jpeg_read_header since
(a) suspension is not possible with the stdio data source, and
(b) we passed TRUE to reject a tables-only JPEG file as an error.
See libjpeg.doc for more info. */
/* Step 4: set parameters for decompression */
/* In this example, we don't need to change any of the defaults set by
jpeg_read_header(), so we do nothing here. */
/* Step 5: Start decompressor */

(void) jpeg_start_decompress(&cinfo);

/* We can ignore the return value since suspension is not possible
with the stdio data source. */
/* We may need to do some setup of our own at this point before reading the data. After
jpeg_start_decompress() we have the correct scaled output image dimensions available, as well as
the output colormap if we asked for color quantization. In this example, we need to make an output
work buffer of the right size. */
/* JSAMPLEs per row in output buffer */

row_stride = cinfo.output_width * cinfo.output_components;

/* Make a one-row-high sample array that will go away when done with image */

buffer = (*cinfo.mem->alloc_sarray)(j_common_ptr) &cinfo, JPOOL_IMAGE, row_stride, 1);

/* Step 6: while (scan lines remain to be read) */

jpeg_read_scanlines(...);

/* Here we use the library's state variable cinfo.output_scanline as the
```

```
loop counter, so that we don't have to keep track ourselves. */

while(cinfo.output_scanline<cinfo.output_height)
{

/* jpeg_read_scanlines expects an array of pointers to scanlines.
Here the array is only one element long, but you could ask for
more than one scanline at a time if that's more convenient. */

        (void) jpeg_read_scanlines(&cinfo, buffer, 1);

/* Assumeput_scanline_someplace wants a pointer and sample count. */

        put_scanline_someplace(buffer[0], row_stride);
}

/* Step 7: Finish decompression */

(void) jpeg_finish_decompress(&cinfo);

/* We can ignore the return value since suspension is not possible
with the stdio data source. */
/* Step 8: Release JPEG decompression object */
/* This is an important step since it will release a good deal of memory. */

jpeg_destroy_decompress(&cinfo);



/* Afterfinish_decompress, we can close the input file.Here we postpone it until after no more JPEG
errors are possible,
so as to simplify the setjmp error logic above. (Actually, I don't
think that jpeg_destroy can do an error exit, but why assume anything...) */

fclose(infile);

/* At this point you may want to check to see whether any corrupt-data
warnings occurred (test whether jerr.pub.num_warnings is nonzero). */
/* And we're done! */

return 1;
}

/* SOME FINE POINTS:
In the above code, we ignored the return value of jpeg_read_scanlines, which is the number of
scanlines actually read. We could get away with this because we asked for only one line at a time
and we weren't using a suspending data source. See libjpeg.doc for more info.
*
We cheated a bit by calling alloc_sarray() after jpeg_start_decompress();
we should have done it beforehand to ensure that the space would be counted against the JPEG
max_memory setting. In some systems the above code would risk an out-of-memory error.
However, in general we don't know the output image dimensions before jpeg_start_decompress(),
unless we call jpeg_calc_output_dimensions(). See libjpeg.doc for more about this.
*
Scanlines are returned in the same order as they appear in the JPEG file,which is standardly top-to-
bottom. If you must emit data bottom-to-top,you can use one of the virtual arrays provided by the
JPEG memory managerto invert the data. See wrbmp.c for an example.
*
As with compression, some operating modes may require temporary files. On some systems you
may need to set up a signal handler to ensure that temporary files are deleted if the program is
interrupted. See libjpeg.doc.
```

Output



```
 Input
48.0    39.0    40.0    68.0    60.0    38.0    50.0    121.0
149.0   82.0    79.0    101.0   113.0   106.0   27.0    62.0
58.0    63.0    77.0    69.0    124.0   107.0   74.0    125.0
80.0    97.0    74.0    54.0    59.0    71.0    91.0    66.0
18.0    34.0    33.0    46.0    64.0    61.0    32.0    37.0
149.0   108.0   80.0    106.0   116.0   61.0    73.0    92.0
211.0   233.0   159.0   88.0    107.0   158.0   161.0   109.0
212.0   104.0   40.0    44.0    71.0    136.0   113.0   66.0



 On DCT
699.2499999999999       43.17537783517842       55.2458962644029        72.11192252561607       24.000000000000046      -25.50537751002191      11.211754811037084      -4.13619348198467
-129.78397283623318     -71.4957835348326       -70.25934416341046      -73.35229687341331      59.432585562295756      -24.0223298746278       22.614760810882366      -2.05066594498004
85.70740770894031       30.32177947343969       61.77691193468116       44.8663846352026        14.844230984030161      17.348004771994926      15.510407640085607      -13.1934898664740
-40.809000772948515     10.166167106402561      -17.52558849075414      -55.805100649235996     30.50000746240999       -2.279391113997457      -21.002352160175203     -1.2567839947412
-157.49999999999997     -49.39469576644716      13.267763314920942      -1.7801066852396144     -8.749999999999988      22.468610979498212      -8.472261916064841      -9.2348129911366
92.4893831808439        -9.0274872527208        45.72009525326298       -48.1304191876491       -58.505400504935224     -9.012918835424449      -28.5397689145109       10.37937054199913
-53.090043923635406     -62.968866490330015     -3.489592359914488      -19.62162006305608      56.08886972022844       -2.2530070591154256     -3.276911934581287      11.9127500233950
-20.538618492594377     -55.90169401863137      -20.586885431597857     -18.190873560968313     -26.57556076417692      -27.068025625951428     8.472705090576747       0.31380301949317



 On Quant
44.0    4.0     5.0     5.0     1.0     -1.0    0.0     0.0
-11.0   -6.0    -6.0    -4.0    2.0     0.0     0.0     0.0
6.0     2.0     4.0     2.0     0.0     0.0     0.0     0.0
-3.0    1.0     -1.0    -2.0    1.0     0.0     0.0     0.0
-9.0    -2.0    0.0     0.0     0.0     0.0     0.0     0.0
4.0     0.0     1.0     -1.0    -1.0    0.0     0.0     0.0
-1.0    -1.0    0.0     0.0     1.0     0.0     0.0     0.0
0.0     -1.0    0.0     0.0     0.0     0.0     0.0     0.0
```

Practical 10

Write a program to find tokens and eliminate stop word from a file.

```
#include <stdio.h>

#include <conio.h>

#include <stdlib.h>

#include <string.h>

#include<ctype.h>

#define BUFFER_SIZE 1000

void removeAll(char * str, const char * toRemove);

int isKeyword(char buffer[])

{         char keywords[32][10] = {"auto", "break", "case", "char", "const", "continue", "default",
"do", "double", "else", "enum", "extern", "float", "for", "goto", "if", "int", "long", "register", "return",
"short", "signed", "sizeof", "static", "struct", "switch", "typedef", "union", "unsigned", "void",
"volatile", "while"};

          int i, flag = 0;

          clrsrc();

          for(i = 0; i < 32; ++i)

          {         if(strcmp(keywords[i], buffer) == 0)

                    {         flag = 1;

                              break;

                    }

          }

          return flag;

}

void main()

{         FILE * fPtr;

          FILE * fTemp;

          FILE * fp;

          char path[100];

          char toRemove[100];

          char buffer[1000];
```

```
	char ch,operators[] = "+-*/%=";
	int i,j=0;
	fp = fopen("program.txt","r");
	if(fp == NULL)
	{	printf("error while opening the file\n");
		exit(0);
	}
	while((ch = fgetc(fp)) != EOF)
	{	for(i = 0; i < 6; ++i)
		{	if(ch == operators[i])
			{	printf("%c is operator\n", ch);    }
		}
		if(isalnum(ch))
		{	buffer[j++] = ch;		}
		else if((ch == ' ' || ch == '\n') && (j != 0))
		{	buffer[j] = '\0';
			j = 0;
			if(isKeyword(buffer) == 1)
				printf("%s is keyword\n", buffer);
			else
				printf("%s is indentifier\n", buffer);
		}
	}
	fclose(fp);
	getch();
/* Input source file path */
	printf("\nEnter path of source file:");
	scanf("%s", path);
	printf("\nEnter word to remove: ");
	scanf("%s", toRemove);
/*  Open files */
```

```
            fPtr  = fopen(path, "r");
            fTemp = fopen("delete.tmp", "w");
/* fopen() return NULL if unable to open file in given mode. */
            if (fPtr == NULL || fTemp == NULL)
            {
/* Unable to open file hence exit */
                        printf("\nUnable to open file.\n");
                        printf("Please check whether file exists, and you have read/write privilege.\n");
                        exit(EXIT_SUCCESS);
            }
/*          * Read line from source file and write to destination
            * file after removing given word.
*/
            while((fgets(buffer, BUFFER_SIZE, fPtr)) != NULL)
            {
// Remove all occurrence of word from current line
                        removeAll(buffer, toRemove);
// Write to temp file
                        fputs(buffer, fTemp);
            }
/* Close all files to release resource */
            fclose(fPtr);
            fclose(fTemp);
/* Delete original source file */
            remove(path);
/* Rename temp file as original file */
            rename("delete.tmp", path);
            printf("\nAll occurrence of '%s' removed successfully.", toRemove);
            getch();
}
/**         *Remove all occurrences of a given word in string.          */
```

```
void removeAll(char * str, const char * toRemove)
{           int i, j, stringLen, toRemoveLen;
            int found;
            stringLen   = strlen(str);  // Length of string
            toRemoveLen = strlen(toRemove); // Length of word to remove
            for(i=0; i <= stringLen - toRemoveLen; i++)
            {
/* Match word with string */
                        found = 1;
                        for(j=0; j < toRemoveLen; j++)
                        {           if(str[i + j] != toRemove[j])
                                    {           found = 0;
                                                break;
                                    }
                        }
/* If it is not a word */
                        if(str[i + j] != ' ' && str[i + j] != '\t' && str[i + j] != '\n' && str[I + j] != '\0')
                        {           found = 0;          }
/*          * If word is found then shift all characters to left
            * and decrement the string length
*/
                        if(found == 1)
                        {           for(j=i; j <= stringLen - toRemoveLen; j++)
                                    {           str[j] = str[j + toRemoveLen];      }
                                    stringLen = stringLen - toRemoveLen;
// We will match next occurrence of word from current index.
                                    i--;
                        }
            }
}
```

Program.txt file

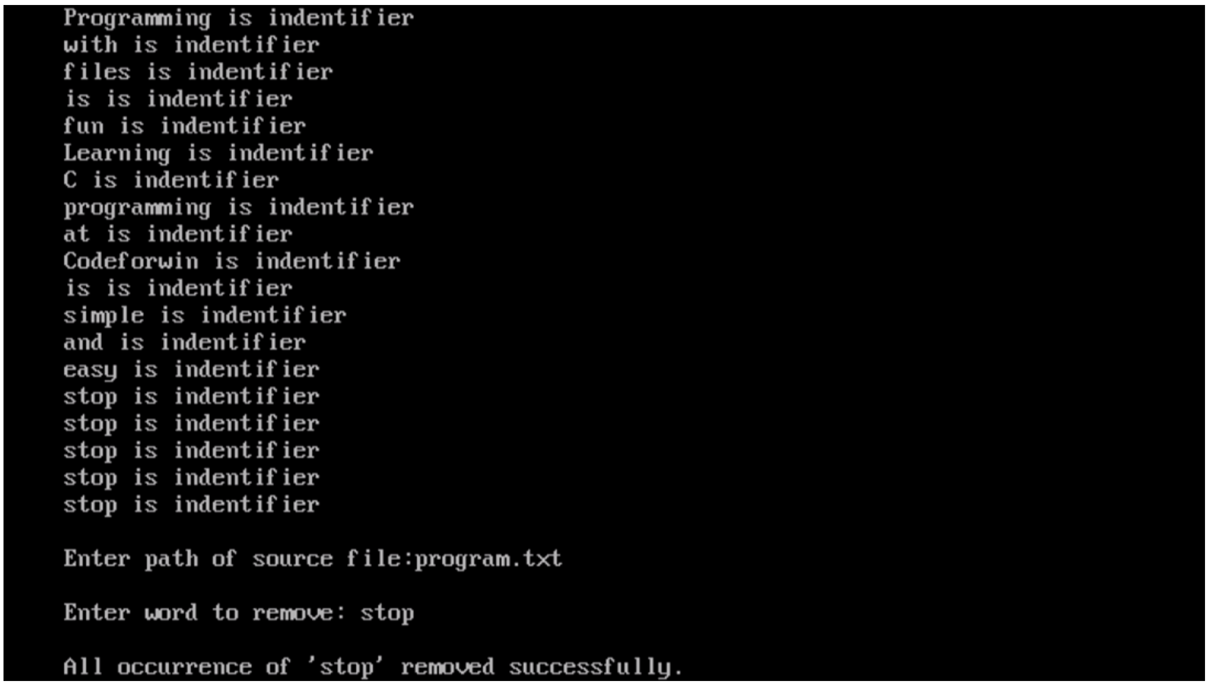
I love programming.

Programming with files is fun.

Learning C programming at Codeforwin is simple and easy.

Stop stop stop stop stop

Output



```
Programming is indentifier
with is indentifier
files is indentifier
is is indentifier
fun is indentifier
Learning is indentifier
C is indentifier
programming is indentifier
at is indentifier
Codeforwin is indentifier
is is indentifier
simple is indentifier
and is indentifier
easy is indentifier
stop is indentifier
stop is indentifier
stop is indentifier
stop is indentifier
stop is indentifier

Enter path of source file:program.txt

Enter word to remove: stop

All occurrence of 'stop' removed successfully.
```