**ANIMATION OF A CIRCLE USING C++ GRAPHICS**

**SOURCE CODE:**

#include<graphics.h>

#include<iostream>

#include<stdlib.h>

#include<iomanip>

#define n 10

void draw(int cx, int cy, int r)

{

inti,gap=0;

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

{

setcolor(rand()%10);

circle(cx,cy,r+gap);

gap+=10;

}

delay(100);

cleardevice();

}

int main()

{

intgdriver=DETECT,gmode;

initgraph(&gdriver, &gmode, "");

int cx=300,cy=200,r=10;

while(1)

{

draw(cx,cy,r);

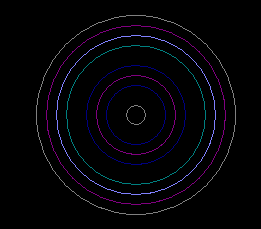
}

getch();

closegraph();

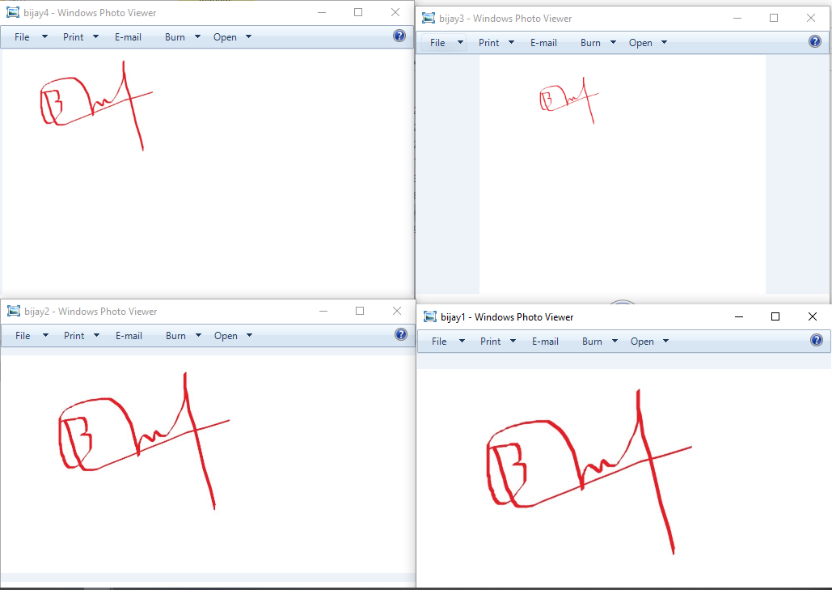
}

**OUTPUT:**

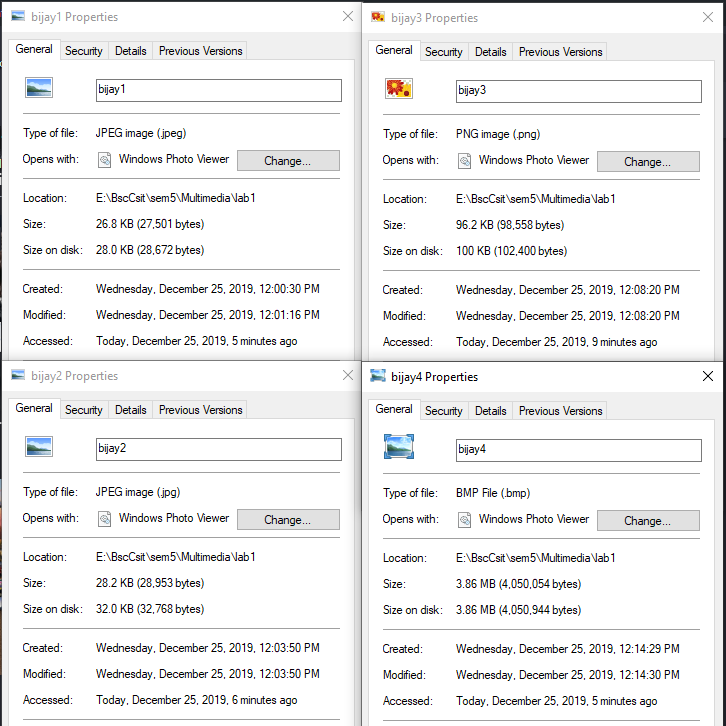


**COMPARE DIFFERENT PROPERTIES OF DIGITAL SIGNATURE ON DIFFERENT FILE FORMAT.**

**DIGITAL SIGNATURE:**

****

**PROPERTIES ON DIFFERENT FILE FORMAT:**



**WRITE A PROGRAM TO IMPLEMENT RUN-LENGTH CODING.**

**SOURCE CODE**

#include <bits/stdc++.h>

using namespace std;

int printRLE(string str)

{

int n = str.length(),count2=0;

for (int i = 0; i < n; i++)

{

int count = 1;

while (i < n - 1 && str[i] == str[i + 1]) {

count++;

i++;

}

cout << str[i]<< count;

count2=count2+2;

}

return count2;

}

return 0;

}

int main()

{

int a,i,p;

string str;

cout<<"Enter the string:";

cin>>str;

i=str.length();

cout<<"\nLength before compression:"<<i;

cout<<"\n\n";

a=printRLE(str);

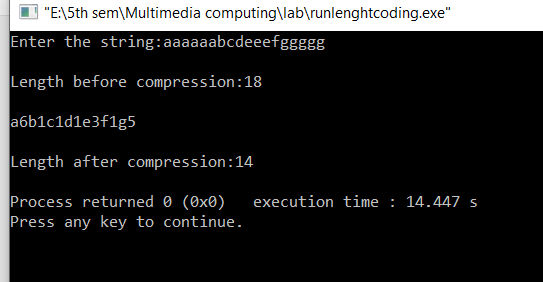
cout<<"\n\nLength after compression:"<<a;

cout<<"\n";

return 0;

}

**Output:**



**OUTPUT:**

**WRITE A PROGRAM TO IMPLEMENT HUFFMAN CODING.**

if (l + 1 < qend && q[l + 1]->freq < q[l]->freq) l++;

q[i] = q[l], i = l;

}

q[i] = q[qend];

return n;

}

/\* walk the tree and put 0s and 1s \*/

void build\_code(node n, char \*s, int len)

{

static char \*out = buf;

if (n->c) {

s[len] = 0;

strcpy(out, s);

code[n->c] = out;

out += len + 1;

return;

}

s[len] = '0'; build\_code(n->left, s, len + 1);

s[len] = '1'; build\_code(n->right, s, len + 1);

}

void init(const char \*s)

{

int i, freq[128] = {0};

char c[16];

while (\*s) freq[(int)\*s++]++;

for (i = 0; i < 128; i++)

if (freq[i]) qinsert(new\_node(freq[i], i, 0, 0));

while (qend > 2)

qinsert(new\_node(0, 0, qremove(), qremove()));

build\_code(q[1], c, 0);

}

void encode(const char \*s, char \*out)

{

while (\*s) {

strcpy(out, code[\*s]);

out += strlen(code[\*s++]);

}

}

**SOURCE CODE:**

#include <stdio.h>

#include <string.h>

typedef struct node\_t {

struct node\_t \*left, \*right;

int freq;

char c;

} \*node;

struct node\_t pool[256] = {{0}};

node qqq[255], \*q = qqq - 1;

int n\_nodes = 0, qend = 1;

char \*code[128] = {0}, buf[1024];

node new\_node(int freq, char c, node a, node b)

{

node n = pool + n\_nodes++;

if (freq) n->c = c, n->freq = freq;

else {

n->left = a, n->right = b;

n->freq = a->freq + b->freq;

}

return n;

}

/\* priority queue \*/

void qinsert(node n)

{

int j, i = qend++;

while ((j = i / 2)) {

if (q[j]->freq <= n->freq) break;

q[i] = q[j], i = j;

}

q[i] = n;

}

node qremove()

{

int i, l;

node n = q[i = 1];

if (qend < 2) return 0;

qend--;

while ((l = i \* 2) < qend) {

void decode(const char \*s, node t)

{

node n = t;

while (\*s) {

if (\*s++ == '0') n = n->left;

else n = n->right;

if (n->c) putchar(n->c), n = t;

}

putchar('\n');

if (t != n) printf("garbage input\n");

}

#define n 1000000

int main()

{

int i,a,b,a1;

char st[n];

char \*str=st;

printf("\nEnter message:");

scanf("%[^\n]s",st);

char buf[1024];

a=strlen(st);

printf("\nTotal message length: %d\n",a);

a1=a\*8;

printf("So, total bits before compressing is %d\*8: %d bits.\n\n",a,a1);

printf("\nBits given for each character for compressing the given message:\n");

init(str);

for (i = 0; i < 128; i++)

if (code[i])

printf("'%c': %s\n", i, code[i]);

encode(str, buf);

b=strlen(buf);

printf("\n\tEncoded/compressed message:%s\n",buf);

printf("\n\tTotal bits after compressing: %d bits.\n",b);

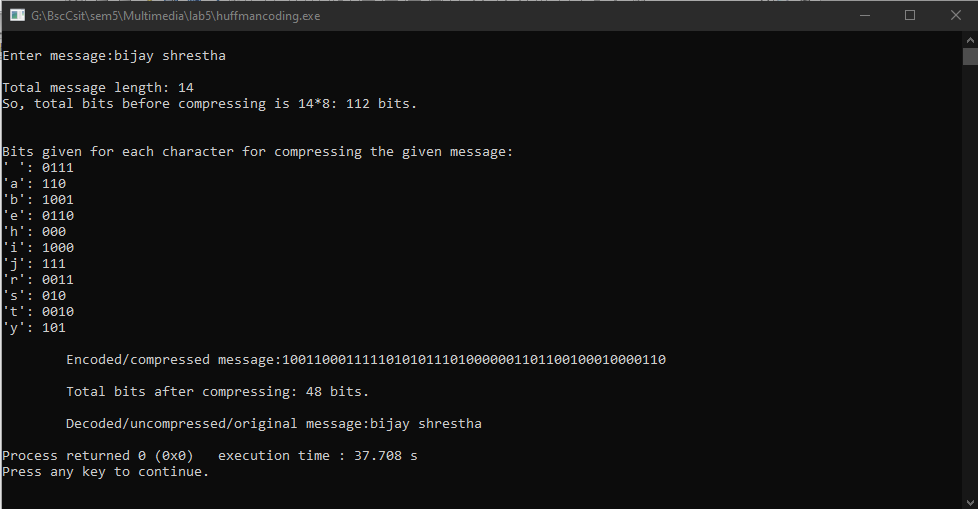
printf("\n\tDecoded/uncompressed/original message:");

decode(buf, q[1]);

return 0;

}

**OUTPUT:**

****

**TO DESIGN A LOGO USING PHOTOSHOP**

