Algorithmique et Programmation

—

Examen sur machine - Solution

—

G1: keriven*(at)*certis.enpc.fr G2: juan*(at)*certis.enpc.fr
G3: gmellier*(at)*melix.org G4: pierre.maurel*(at)*ens.fr
G5: pierre*(at)*senellart.com G6: adde*(at)*certis.enpc.fr

—

07/01/05

—

1 Solution

1.1 Calcul de l'exponentielle d'un nombre complexe

Fichier `complexe.h`

```
1   #pragma once
2   struct complexe
3   {
4       double reel;
5       double imag;
6   };
7   complexe operator*(complexe z1, complexe z2);
8   complexe operator+(complexe z1, complexe z2);
9   complexe operator/(complexe z, double d);
```

Fichier `complexe.cpp`

```
1   #include "complexe.h"
2   complexe operator*(complexe z1, complexe z2)
3   {
4       complexe z;
5       z.reel=z1.reel*z2.reel-z1.imag*z2.imag;
6       z.imag=z1.reel*z2.imag+z1.imag*z2.reel;
7       return z;
8   }
9   complexe operator+(complexe z1, complexe z2)
10  {
11      complexe z;
12      z.reel=z1.reel+z2.reel;
13      z.imag=z2.imag+z1.imag;
14      return z;
15  }
16  complexe operator/(complexe z, double d)
17  {
18      complexe r;
19      r.reel=z.reel/d;
20      r.imag=z.imag/d;
21      return r;
22  }
```

Fichier `main.cpp`

```
#include <iostream>
#define _USE_MATH_DEFINES
#include <cmath>
using namespace std;
#include "complexe.h"

complexe exponentielle(complexe z, int n)
{
    complexe zn={1,0},e=zn;
    double fact=1;
    for(int i=1;i<(n+1);i++)
    {
        fact  = fact*i;
        zn = zn*z;
        e = e+ zn/fact;
    }
    return e;
}
void cos_sin(double theta, int n, double& cos, double& sin)
{
    complexe z={0,theta};
    complexe e=exponentielle(z,n);
    cos = e.reel;
    sin = e.imag;
}
int main()
{
    double cos,sin;
    double theta=M_PI/6;
    int n=15;
    cos_sin(theta,n,cos,sin);
    cout<<"le cosinus vaut "<<cos<<endl;
    cout<<"le sinus vaut "<<sin<<endl;
    return 0;
}
```

1.2 Compression RLE

```
// Exemple Winlib5
#include <iostream>
using namespace std;
#include <win>
using namespace Win;

void affiche_image(byte image[],int w, int h) {
    PutGreyImage(0,0,image,w,h);
}
void remplir_rectangle(byte image[], int w, int h,int xul, int yul, int height, int width ) {
    for (int i=xul;(i<xul+width)&&(i<w);i++)
        for (int j=yul;(j<yul+height)&&(j<h);j++)
            image[i+j*w]=byte(255);
}
void RLE_encode(byte source_image[], int w, int h,int compression[], int &comp_size ) {
    int count;
    byte value=0;
    int pos_source=0;
    comp_size=0;
    do {
        count=0;
        while ((pos_source<w*h)&&(source_image[pos_source]==value)){
            count++;
```

```
24              pos_source++;
25          }
26          compression[comp_size]=count;
27          comp_size++;
28          if(value==0)value=255;
29          else value=0;
30      }while(pos_source<w*h);
31  }
32  void RLE_decode( int compression[], int comp_size,byte decomp_image[]) {
33      int count=0;
34      byte value=0;
35      for (int i=0;i<comp_size;i++) {
36          for (int j=0;j<compression[i];j++,count++) {
37              decomp_image[count]=value;
38          }
39          if(value==0)value=255;
40          else value=0;
41      }
42  }
43  void remplir_diagonale( byte image[], int w, int h ) {
44      int borne_sup=min(w,h);
45      for (int i=0;i<borne_sup;i++) {
46          image[i*(w+1)]=byte(255);
47      }
48  }
49  int main()
50  {
51      //constantes
52      const int w=1024,h=1024;
53      //taille de l'image une fois compressée
54      int comp_size;
55      //allocation statique (question 2)
56      //byte image_source[w*h];
57      //byte image_decodee[w*h];
58      //int image_encodee[w*h];
59      //allocation dynamique (question 13)
60      byte* image_source=new byte[w*h];
61      byte* image_decodee=new byte[w*h];
62      int* image_encodee=new int[w*h+1];
63
64      //ouverture d'une fenetre et affichage d'une image non initialisée (question 4)
65      OpenWindow(w,h);
66      affiche_image(image_source,w,h);
67
68      Click();
69      //initialisation, remplissage et affichage de l'image source (question 6)
70      for (int i=0;i<w*h;i++)
71          image_source[i]=byte(0);
72      remplir_rectangle(image_source,w,h,50,50,70,20);
73      affiche_image(image_source,w,h);
74
75      //reinitialisation, remplissage, affichage et compression (question 9)
76      for (int i=0;i<w*h;i++)
77          image_source[i]=byte(0);
78      remplir_rectangle(image_source,w,h,50,50,70,20);
79      remplir_rectangle(image_source,w,h,20,30,25,90);
80      affiche_image(image_source,w,h);
81
82      Click();
83      //compression
84      RLE_encode(image_source,w,h,image_encodee,comp_size);
```

```
85      //decompression
86      RLE_decode(image_encodee,comp_size,image_decodee);
87
88      //verification
89      bool good=true;
90      int pos=0;
91      do {
92          good=(image_source[pos]==image_decodee[pos]);
93          pos++;
94      } while((pos<w*h)&&(good));
95      if (!good)cout<<"l'image source et decodee ne sont pas identiques !"<<endl;
96
97      affiche_image(image_decodee,w,h);
98      cout<<comp_size<<endl;
99
100     Click();
101     //reinitialisation, remplissage et affichage de l'image source (question 11)
102     for (int i=0;i<w*h;i++)
103         image_source[i]=byte(0);
104     remplir_diagonale(image_source,w,h);
105     affiche_image(image_source,w,h);
106
107     Click();
108     //reinitialisation, remplissage et affichage de l'image source (question 12)
109     for (int i=0;i<w*h;i++)
110         image_source[i]=byte(0);
111     Click();
112     remplir_diagonale(image_source,w,h);
113     remplir_rectangle(image_source,w,h,20,20,80,80);
114     remplir_rectangle(image_source,w,h,120,10,100,100);
115     affiche_image(image_source,w,h);
116
117     //compression
118     RLE_encode(image_source,w,h,image_encodee,comp_size);
119     cout<<comp_size<<endl;
120
121     Click();
122     //reinitialisation, remplissage et affichage de l'image source (question 12 bis)
123     for (int i=0;i<w*h;i++)
124         image_source[i]=byte(0);
125     remplir_diagonale(image_source,w,h);
126     affiche_image(image_source,w,h);
127
128     //compression
129     RLE_encode(image_source,w,h,image_encodee,comp_size);
130     cout<<comp_size<<endl;
131     affiche_image(image_source,w,h);
132
133     Terminate();
134
135     //desallocation (question 13)
136     delete[] image_source;
137     delete[] image_encodee;
138     delete[] image_decodee;
139
140     return 0;
141 }
```