UNIVERSIDAD NACIONAL AUTÓNOMA DE HONDURAS
ESCUELA DE MATEMÁTICAS
EXAMEN DEL SEGUNDO PARCIAL

MM-418 Programación II **I Período 2018** **miércoles 11 de abril**

Nombre:_____ PAUTA _____ No. de Cuenta:_____

Profesor: **David Motiño** Sección: **800** No. Lista:_____

Instrucciones: Resuelva de forma clara y ordenada los siguientes ejercicios. Escribir la codificación en el lenguaje de programación C/C++.

(30$^{\text{Pts}}$) **1. PROBLEMA 1:**

30 Pts

Implementar una plantilla de la **clase complejo**, la cual tiene como atributos:

- La parte real e imaginaria: Dos datos genéricos.

Para esta clase debe implementar las siguiente funciones:

1. **El constructor alternativo**: Recibe dos números que corresponden a la parte real e imaginaria.**(5%)**

2. **Sobrecarga del operador suma** (+) **(5%)**

3. la función **módulo** del número complejo. **(5%)**
4. La función **conjugado**.**(5%)**

5. **Sobrecarga del operador** (<<): para imprimir el número complejo de la forma: $a + bi$. **(5%)**

6. Crear un número complejo con parte real e imaginaria entera y un número complejo con parte real e imaginaria booleana, imprimir dichos números complejos y sus módulos.**(5%)**

Listing 1: Plantilla de clase complejo

```
1  #include <iostream>
2  #include <cmath>
3  using namespace std;
4  template<class T>
5  class complejo{
6  friend ostream&operator<<(ostream&escribir ,complejo<T>&z){
7  if(z.imaginario<0)
8  escribir<<z.real<<z.imaginario<<"i\n";
9  else
10 escribir<<z.real<<"+"<<z.imaginario<<"i\n";
11 return escribir;
12 }
13 private:
14 T real;
15 T imaginario;
16 public:
17 complejo(T,T);
18 complejo<T>operator+(const complejo<T>&)const;
19 float modulo()const;
20 complejo<T>conjugado()const;
21
22 };
23 int main(int argc, char** argv) {
24 complejo<int>z1(1,-2);
25 complejo<float>z2(1.33,sqrt(2));
26 cout<<z1<<endl;
27 cout<<z2<<endl;
28 cout<<"Modulo de z1="<<z1.modulo()<<endl;
29 cout<<"Modulo de z2="<<z2.modulo()<<endl;
30 return 0;
31 }
32
33 template<class T>
34 complejo<T>::complejo(T x,T y){
35 real=x;
36 imaginario=y;
37 }
38
39 template<class T>
40 complejo<T> complejo<T>::operator+(const complejo<T>&z)const{
41 complejo<T> suma(0,0);
42 suma.real=real+z.real;
43 suma.imaginario=imaginario+z.imaginario;
44
45 return suma;
46 }
47
48 template<class T>
49 float complejo<T>::modulo()const{
50
51 return sqrt(pow(real,2)+pow(imaginario,2));
52 }
53
54 template<class T>
55 complejo<T>complejo<T>::conjugado()const{
56 complejo<T> c(real,-imaginario);
57 return c;
58 }
```

(40 Pts) **2. PROBLEMA 2:**

40 Pts

Implementar herencia entre la clase base **tridiagonal** y la clase derivada **diagonal**. Aplique **polimorfismo** para las funciones miembro comunes en dichas clases.

Clase tridiagonal

Tienen como atributos:

- El orden de la matriz: **n**
- Los elementos de la matriz: **elementos**
  Nota: Debe utilizarse una estructura de almacenamiento (**usar memoria dinámica**) que almacene sólo los elementos de la matriz que están en la diagonal, en la diagonal superior e inferior.

Para esta clase debe implementar las siguiente funciones:

1. **El constructor alternativo (5%)**: Recibe el orden de la matriz y genera los elementos de forma aleatoria.

2. **El destructor (2%)**

3. **La traza(3%)**: Es la suma de todos los elementos de la diagonal.

4. **Funcioń imprimir (5%)**

Clase diagonal
Tiene como atributos:

- El orden de la matriz: **n**
- Los elementos de la matriz: **elementos**

Nota: Debe utilizarse una estructura de almacenamiento (**usar memoria dinámica**) que almacene sólo los elementos de la diagonal.

Para esta clase debe implementar las siguiente funciones:

1. **El constructor alternativo(3%)**: Recibe el orden de la matriz y genera los elementos de forma aleatoria.

2. **La traza(2%)**: Es la suma de todos los elementos de la diagonal.

3. **Función imprimir (5%)**

4. **El determinante de la matriz (5%)**

5. **En el main:** Crear un arreglo de tamaño 5 de apuntadores a objetos de la clase tridiagonal, donde los primero tres elementos del arreglo deben de apuntar a objetos de la clase **tridiagonal** y los demás deben de apuntar a ojetos de la clase **diagonal**. Mediante el arreglo de apuntadores debe de imprimir las 5 matrices y debe de calcular su traza. **(10%)**

Listing 2: Herencia: tridiagonal-diagonal

```
1  #include <iostream>
2  #include <cstdlib>
3  #include <ctime>
4  #include <iomanip>
5  using namespace std;
6
7  class tridiagonal{
8
9  protected:
10 int n;
11 double *dia;
12 double *diaInf;
13 double *diaSup;
14 public:
15 tridiagonal();
16 tridiagonal(int);
17 ~tridiagonal();
18 virtual double traza()const;
19 virtual void imprimir()const;
20 };
21
22 class diagonal:public tridiagonal{
23
24 public:
25 diagonal(int);
26 ~diagonal();
27 virtual double traza()const;
28 virtual void imprimir()const;
29
30 };
31 int main(int argc, char** argv) {
32 tridiagonal**A=new tridiagonal*[5];
33 A[0]=new tridiagonal(5);
34 A[1]=new tridiagonal(6);
35 A[2]=new tridiagonal(10);
36 A[3]=new diagonal(3);
37 A[3]=new diagonal(6);
38 for(int i=0;i<5;i++)
39 {A[i]->imprimir();
40 cout<<"Traza="<<A[i]->traza()<<endl;
41 }
42 for(int i=0;i<5;i++)
43 delete A[i];
44 delete[]A;
45 return 0;
46 }
47
48 /*################CLASE TRIDIAGONAL###############*/
49
50 tridiagonal::tridiagonal(){
51 n=2;
52 dia=new double[2];
53 diaSup=new double[1];
54 diaInf=new double[1];
55 dia[0]=1;
56 dia[1]=1;
57 diaSup[0]=1;
58 diaInf[0]=1;
```

```
59 }
60 tridiagonal::tridiagonal(int tam){
61 n=tam;
62 dia=new double[n];
63 diaInf=new double[n-1];
64 diaSup=new double[n-1];
65 for(int i=0;i<n-1;i++){
66 dia[i]=rand()%10;
67 diaInf[i]=rand()%10;
68 diaSup[i]=rand()%10;
69 }
70 dia[n-1]=rand()%10;
71 }
72
73 tridiagonal::~tridiagonal(){
74 delete[]dia;
75 delete[]diaInf;
76 delete[]diaSup;
77 }
78
79 double tridiagonal::traza()const{
80 double s=0;
81 for(int i=0;i<n;i++)
82 s+=dia[i];
83 return s;
84 }
85
86 void tridiagonal::imprimir()const{
87 cout<<"\n#######################################\n";
88 for(int i=0;i<n;i++)
89 {for(int j=0;j<n;j++)
90 if(i==j)
91 cout<<setw(3)<<dia[i];
92 else
93 if(i-j==1)
94 cout<<setw(3)<<diaInf[j];
95 else
96 if(i-j==-1)
97 cout<<setw(3)<<diaSup[i];
98 else
99 cout<<setw(3)<<0;
100 cout<<endl;
101 }
102 cout<<endl;
103 }
104
105 /*###################CLASE DIAGONAL###############*/
106 diagonal::diagonal(int tam){
107 n=tam;
108 dia=new double[n];
109 for(int i=0;i<n;i++){
110 dia[i]=rand()%10;
111 }
112 }
113
114 diagonal::~diagonal(){
115 delete[]dia;
116 }
117
```

```
118 double diagonal::traza()const{
119 double s=0;
120 for(int i=0;i<n;i++)
121 s+=dia[i];
122 return s;
123 }
124
125 void diagonal::imprimir()const{
126 cout<<"\n########################################\n";
127 for(int i=0;i<n;i++){
128 for(int j=0;j<n;j++)
129 if(i==j)
130 cout<<setw(4)<<dia[i];
131 else
132 cout<<setw(4)<<0;
133 cout<<endl;
134 }
135 cout<<endl;
136 }
```