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

MM-418 Programación II **I Período 2018** **viernes 2 de marzo**

Nombre:_____ PAUTA _____ No. de Cuenta:_____

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

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

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

30 Pts

Para la **clase matriz**, su interfaz y las definiciones de algunas funciones miembro y no miembro se muestran a constinuación.
Indique qué instrucciones son incorrectas o están imcompletas, escriba en dichos casos las instruciones de forma correcta o complete el bloque de código incompleto según sea el caso.

Listing 1: Clase matriz

```
1  class matriz{
2  friend matriz operator-(const matriz&);
3  friend matriz operator--(matriz&)const;//Posdecremento
4  bool operator<(const matriz&,const matriz&);
5
6  private:
7  int NumFilas, NumColumnas;
8  float**elementos;
9
10 public:
11 matriz();
12
13 matriz(int,int){
14
15 for(int i=0,i<NumFilas;i++)
16 for(int j=0,j<Numcolumnas;j++)
17      elementos[i][j]=rand()%10;
18 }
19
20 ~matriz(){
21 delete elementos;
22 }
23
24 void imprimir(const matriz&)const;
25 matriz  suma(const matriz &)const;//Suma de matrices.
26 matriz  operator*(const matriz &, const matriz&)const;//Multiplicacion de
27 matriz operator*(float , matriz&);//Multiplicacion de una matriz por un esc
28 matriz operator+(const matriz&)const;
29 matriz operator++(int);//preincremento
30 ostream& operator<<(ostream&,const matriz&);
31 matriz operator==(const matriz&)const;
32 bool operator>(const matriz&)const;
33
34 };
```

Solución

Listing 2: Clase Matriz

```
1   class matriz{
2   friend matriz operator-(const matriz&,const matriz&); //CORREGIDO
3   friend matriz operator--(matriz&, int)const;//Posdecremento//CORREGIDO
4   friend bool operator<(const matriz&,const matriz&);//CORREGIDO
5
6   private:
7   int NumFilas, NumColumnas;
8   float**elementos;
9
10  public:
11  matriz();
12
13  matriz(int n,int m){
14  //AGREGANDO BLOQUE QUE FALTABA
15  /*####################################*/
16  NumFilas=n;
17  NumColumnas=m;
18  elementos=new float*[n];
19  for(int i=0,i<NumFilas;i++)
20  elementos[i]=new float[m];
21  /*####################################*/
22  for(int i=0,i<NumFilas;i++)
23  for(int j=0,j<Numcolumnas;j++)
24  elementos[i][j]=rand()%10;
25  }
26
27  ~matriz(){
28      //AGREGANDO BLOQUE QUE FALTABA
29  /*####################################*/
30  for(int i=0;i<NumFilas;i++)
31      delete[]elementos[i];
32  /*####################################*/
33  delete[]elementos;
34  }
35
36  void imprimir()const;//CORREGIDO
37  matriz  suma(const matriz &)const;//Suma de matrices.//CORREGIDO
38  matriz  operator*(const matriz &)const;//Multiplicacion de matrices.
39  friend matriz operator*(float, matriz&);//Multiplicacion de una matriz
40  //por un escalar.//CORREGIDO
41  matriz operator+(const matriz&)const;
42  matriz operator++();//preincremento//CORREGIDO
43  friend ostream& operator<<(ostream&,const matriz&);//CORREGIDO
44  bool operator==(const matriz&)const;//CORREGIDO
45  bool operator>(const matriz&)const;
46
47  };
```

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

40 Pts

Implementar la **clase conjunto**, que consiste en conjuntos finitos de números enteros, esta clase tiene como atributos:

- La cardinalidad (El número de elementos que tiene el conjunto): **n**
- Los elementos del conjunto: **elementos**
  Nota: Utilizar arreglos unidimensionales (**usar memoria dinámica**) para almacenar los elementos del conjunto.

Implementar las siguientes funciones minembros y no miembros:

1. **El constructor alternativo(5%)**: Recibe la cardinalidad del conjunto y genera los elementos de forma aleatoria, en teoría los elementos no deben repetirse, sin embargo, en la implementación de este constructor podrían haber elementos repetidos.

2. **El destructor (5%)**

3. **Imprimir (<<)(5%)**

4. **Función pertenece(int x)(5%)**: Esta función retorna **true** si el conunto contiene el elemento **x** que se manda como parámetro, en caso contrario, retorna **false**.

5. **Unión (+) (5%)**

6. **Postincremento (++) (5%)**: Aumenta todos los elementos del conjunto en una unidad.

7. **La intersección (*) (5%)**: Es el conjunto de todos los elementos que ambos conjunto tienen en común.

8. **Comparación (==)(5%):** Este operador determina si los conjuntos son iguales (Asuma que no hay elementos repetidos en los conjuntos).
   Sugerencia: Primero compare la cardinalidad de los conjuntos, si tienen la misma cardinalidad es posible que los conjuntos tengan los mismos elementos, para ello, ordene ambos arreglos de forma ascendente y compare elemento a elemento.

9. **BONO: El constructor alternativo(5%) sin elementos repetidos**: Recibe la cardinalidad del conjunto y genera los elementos de forma aleatoria pero sin elementos repetidos.

Solución

Listing 3: Clase Conjunto

```
#include <iostream>
#include <cstdlib>
#include <cmath>
#include <ctime>
using namespace std;
/* run this program using the console pauser or add your own getch, system("p
class conjunto{
friend ostream& operator<<(ostream &,const conjunto &);
friend conjunto operator++(const conjunto&,int);
friend conjunto operator*(const conjunto&,const conjunto&);
private:
int n;
int*elementos;

public:

conjunto(int);
~conjunto();
bool pertenece(int)const;
conjunto operator+(const conjunto&)const;
bool operator==( conjunto&);
void burbuja();

};


int main(int argc, char** argv) {

return 0;
}

/*conjunto::conjunto(int card){
n=card;
elementos=new int[n];
for(int i=0;i<n;i++)
elementos[i]=pow(-1,(rand()%2))*(rand()%100);
}*/

conjunto::~conjunto(){
delete[]elementos;
}

ostream& operator<<(ostream & escribir,const conjunto& A){
escribir<<"{";
for(int i=0;i<(A.n)-1;i++)
escribir<<A.elementos[i]<<", ";

escribir<<A.elementos[A.n-1]<<"}";

return escribir;
}

bool conjunto::pertenece(int x)const{
bool resp=false;
for(int i=0;i<n;i++){
if(elementos[i]==x){
```

```
57 resp=true;
58 break;
59 }
60 return resp;
61
62 }
63 }
64
65 conjunto conjunto::operator+(const conjunto&B)const{
66 conjunto c(n+B.n);
67 for(int i=0;i<n;i++)
68 c.elementos[i]=elementos[i];
69
70 for(int i=0;i<B.n;i++)
71 c.elementos[n+i]=B.elementos[i];
72
73 return c;
74 }
75
76 conjunto operator++(const conjunto&A,int m){
77 for(int i=0;i<A.n;i++)
78 A.elementos[i]++;
79
80 return A;
81 }
82
83 conjunto operator*(const conjunto&A,const conjunto&B){
84 int k=0;
85 for(int i=0;i<A.n;i++)
86 for(int j=0;j<B.n;j++)
87 if(A.elementos[i]==B.elementos[j])
88 k++;
89 conjunto c(k);
90 int r=0;
91 for(int i=0;i<A.n;i++)
92 for(int j=0;j<B.n;j++)
93 if(A.elementos[i]==B.elementos[j])
94 {c.elementos[r]=A.elementos[i];
95 r++;
96 }
97
98 return c;
99 }
100
101 bool conjunto::operator==( conjunto&B){
102 if(n!=B.n)
103 return false;
104 else{
105 bool resp=true;
106 this->burbuja();
107 B.burbuja();
108 for(int i=0;i<n;i++)
109 if(elementos[i]!=B.elementos[i]){
110 resp=false;
111 break;
112 }
113 return resp;
114 }
115
```

```
116 }
117
118 conjunto::conjunto(int card){
119 n=card;
120 elementos=new int[n];
121 elementos[0]=pow(-1,(rand()%2))*(rand()%n);
122 bool resp;
123 for(int i=1;i<n;i++)
124 {    do{resp=false;
125 elementos[i]=pow(-1,(rand()%2))*(rand()%n);
126 for(int j=i-1;j>=0;j--)
127 if(elementos[i]==elementos[j])
128 {resp=true;
129 break;
130 }
131
132 }while(resp);
133
134 }
135
136
137 }
138
139 void conjunto::burbuja(){
140 int temp;
141 for(int pasada=1;pasada<n-1;pasada++)
142 for(int i=0;i<n-pasada;i++)
143 if(elementos[i]>elementos[i+1]){
144 temp=elementos[i];
145 elementos[i]=elementos[i+1];
146 elementos[i+1]=temp;
147 }
148
149 }
```