Primer Certamen 1er. Sem 2003

1. Haga el diseño e implementación de una clase que permita cubrir las siguientes demandas:

Deseamos poder crear conjuntos de enteros dentro de nuestro programa. Las operaciones que desarrollaremos con conjunto serán la Unión, la intersección y una función que nos permita determinar si el conjunto está vacío.

No sabemos qué tan grande puede ser nuestro conjunto y se nos ha pedido no usar la STL porque se pretende reutilizar esta clase en sistemas autónomos (hardware dedicado basado en microprocesadores).

Los programadores que usarán su clase le piden que use los operadores + para la unión y * para la intersección.

Incluya: archivo conjunto.h y archivo conjunto.cpp.

conjunto.h

```
#ifndef CONJUNTO_H
#define CONJUNTO_H
class Conjunto {
public:
        Conjunto();
        Conjunto(int elemento); // no es estrictamente necesario
        Conjunto operator + (const Conjunto & conj);
        Conjunto operator * (const Conjunto & conj);
        bool esVacio();
        // opto por incorporar esta función para agregar elementos unitarios a un
conjunto.
        void agregarElemento +(int elemento);
        bool pertenece (int);
        ~Conjunto();
private:
        int * miembros;
        int tamano;
};
#endif
```

conjunto.cpp

```
#include "conjunto.h"
Conjunto::Conjunto():miembros(NULL), tamano(0){}
Conjunto::Conjunto(int elemento){ // no es estrictamente necesario
        miembros = new int[1];
        mienbros[0]=elemento;
        tamano = 1;
}

Conjunto::~Conjunto() {
        if (mienbros !=NULL)
                delete miembros[];
```

```
}

bool Conjunto::pertenece(int e) {
        for (int i=0; i<tamano; i++)
                if (e==miembros[i]) return true;
        return false;
}
void Conjunto::agregarElemento(int e){
        int * tmp, i;
        if (pertenece(e))
                return; // ya está
        tmp = miembros;
        miembros=new int[tamano+1];
        for( i=0; i<tamano; i++)
                miembros[i]=tmp[i];
        miembros[tamano++]=e;
        if (tmp !=NULL) delete tmp[];
}

Conjunto Conjunto::operador +(const Conjunto & conj){
        int i; Conjunto union;
        for (i=0; i<tamano; i++) {
                union.agregarElemento(miembros[i]);
        for (i=0; i<conj.tamano; i++) {
                union.agregarElemento(conj.miembros[i]);
        return union;
}

Conjunto Conjunto::operator * (const Conjunto &conj) {
        int i; Conjunto inter;
        for (i=0; i<tamano; i++)
                if (conj.pertenece(mienbros[i]) ) inter.agregarElemento(miembro[i]);
        return inter;
}

bool Conjunto::esVacio () {
        return tamano==0;
}
```

2. Un empleado de la empresa era el responsable de desarrollar una clase para manejar números enteros que trabajen con aritmética modular. Esto es, si m es el módulo, todos los resultados terminan en algún valor en el rango 0..m-1. Donde m es un parámetro de la case. Este empleado fue “tentado” por la competencia y dejó el trabajo. Usted retoma el tema y encuentra el siguiente archivo.

```
class intmod {
public:
```

```
        intmod();
        intmod (int m);
        intmod operator + (intmod n);
        intmod operator * (intmod n);
        intmod operator +(int n);
        static bool esPar(intmod a);

private
        int i;
        const int m;
}
```

Usted observa la ausencia de una operación que permita enviar una instancia de intmod a un stream de salida.
Complete la clase intmod, eventualmente corríjala y presente su implementación en intmod.cpp.
Si hay cosas que no estén claras, escríbalas en su respuesta. Éstas serían presentadas en su siguiente reunión de grupo de trabajo de su empresa.

Agregamos operador para salida a stream.

```
class intmod {
public:
        intmod();
        intmod (int m);
        intmod operator + (intmod n);
        intmod operator * (intmod n);
        intmod operator +(int n);
        static bool esPar(intmod a);
        friend ostream operator<< (ostream &os, const intmode& n) {
                os << n.i;
        }
private
        int i;
        const int m;
}
```

El resto de la implementaciones sería:

intmod.cpp

```
#include “intmod.h”
intmod::intmod() m(1) {}
intmod::intmod(int mm): i(0), m(mm){}
intmod intmod::operator + (intmod n) {
        // problema a presentar en proxima reunion: Qué pasa si los módulos son
distintos?
        intmod tmp(m);
        tmp.i= (i+n.i)%m;
```

```
        return tmp;
}

intmod intmod::operator * (intmod n) {
        // problema a presentar en proxima reunion: Qué pasa si los módulos son
distintos?
        intmod tmp(m);
        tmp.i= (i*n.i)%m;
        return tmp;
}

intmod intmod::operator + (int n) {
        intmod tmp(m);
        tmp.i= (i+n)%m;
        return tmp;
}

bool intmod::esPar(intmod n) {
        return(n%2 ==0);
}
```

El aspecto no claro es qué pasa con las operaciones entre intmods de distinto módulo.

3. Se tiene la siguiente función para conversión a letras mayúsculas:

```
void Amayuscula(char c) {
        if ((c >=`a`') && (c <= `z`))
                c = `A` + c-`a`;
}
```

Se detectan dos problemas que deben ser corregidos. Uno de ellos es la ausencia de un indicador de excepción. Se desea evidenciar a quien llame esta función cuando el carácter sea no alfabético. Además se le debe indicar, en este caso, cuál fue el carácter específico ingresado como parámetro.

a) ¿Cuál cree usted que fue el otro error?

La idea de llevar a mayúscula no se cumple por tener un parámetro pasado por valor. Se debe cambiar a referencia. Quedaría así:

```
void Amayuscula(char &c) {
        if ((c >=`a`') && (c <= `z`))
                c = `A` + c-`a`;
}
```

b) Haga los cambios para que señalar al código llamador el ingreso de caracteres no alfanuméricos *(Debió decir alfabético, error de enunciado, no crítico en corrección).*

Habría que crear una clase para señalar la excepción.

```
class noAlfabetico {
public:
        noAlfabetico(char ch):c(ch){};
        char getChar() {return c;};
```

```
private
        char c;
};
```

Luego habría que cambiar la función a:

```
void Amayuscula(char &c) {
        if ((c >=`a`) && (c <= `z`))
                c = `A` + c-`a`;
        if ((c < `A`) || (c > `Z`)) throw noAlfabetico(c);
}
```

4.a) Desarrolle una template o plantilla que permita calcular el valor máximo entre los elementos de un arreglo. Como parámetros se ingresarán el arreglo y su tamaño.

```
template <class T>
T maximo(T A[], int n){
        T max=A[0];
        for (int i=1; i<n; i++)
                if (A[i]>max) max=A[i];
        return max;
}
```

b) ¿Podría usar esta plantilla ~~para~~ con un arreglo de intmod según su definición de la pregunta previa? Explique. Si no fuera posible, implemente la modificación requerida.

No se puede usar tal como está.
Para que esta operación pueda ser usada la comparación "mayor que" debe estar definida en la clase intmod.

En la declaración de la clase se debe agregar la operación:

```
public: bool operator > (const intmod & n);
```

y en la implementación de la clase, agregar:

```
bool intmod::operator > (const intmod & n) {
        return (i > n.i);
}
```

5. Sea:

```
class A {
public:
        void display() { cout << "Yo soy A \n";};
};
class B: public A {
        void display() {cout << "Yo soy B \n";};
};
void foo1(A & a) {
        a.display();
```

```
}
void foo2(A a) {
        a.display();
}
void foo3(A * a) {
        a->display();
}
```

a) ¿Cuál será la salida en las siguientes situaciones?
a1) B b; foo1(b); a2) B b; foo2(b); a3) B b; foo3(&b);
a1) Yo soy A
a2) Yo soy A
a3) Yo soy A

b) Si la clase A la cambiamos por

```
class A {
public:
        virtual void display() { cout << "Yo soy A \n";};
};
```

¿Cuál será la salida en las siguientes situaciones?
b1) B b; foo1(b); b2) B b; foo2(b); b3) B b; foo3(&b);
b1) Yo soy B
b2) Yo soy A
b3) Yo soy B