TP N°10: Introducción al C++

Notas respecto a la forma de realizar los ejercicios:

- La definición de la clase se debe realizar en un .h, mientras que la implementación de la misma se debe realizar en el punto cpp (o extensión similar), excepto se solicite lo contrario.
- Los ejercicios, son evolutivos y en cada serie, un ejercicio depende del ejercicio anterior.
- El código debe ser siempre el menor posible. Esto significa que si una porción de código se repite, realizar una macro o encapsularlo en una función. De igual manera si el constructor o destructor por omisión son suficientes, no implementarlos. En todos los casos analizar si la codificación realizada es realmente necesaria.
- Siempre que sea posible, aunque el enunciado no lo exprese, utilizar el atributo **const**.
- Dentro de cada punto, se acompaña el código fuente de la aplicación, y la correspondiente salida en pantalla.
- En el ejercicio 10.1 se puede ver la Macro `PRESS_KEY`, la cual es utilizada en el resto de los ejercicios.

Ejercicio 10.1

Definir e implementar una clase Punto, que posea los elementos públicos y privados:

Variables miembro privadas:
mx y **my** - del tipo **double**, que representan las coordenadas (x; y) del punto.

Métodos miembro públicos:
setPunto: recibe los valores de **x** e **y** en dos variables **double**.
getPunto: devuelve el valor del punto (en formato class Punto). En formato **inline**
setX y **setY**; que dan valor a x e y. En formato inline
getX y **getY**; que devuelven los valores de x e y. En formato **inline**

La creación del objeto debe permitir o no, las siguientes expresiones:
Punto pa; // **mx** y **my** se inicializan en cero.
Punto pb (23.3,56.8); // **mx** se inicializa con 23.3 y **my** con 56.8.

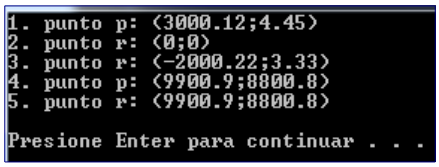
La inicialización del objeto con un solo parámetro debe dar error de sintaxis, por ejemplo la línea:
Punto pc (34.4); // error de sintaxis.

Código Fuente:

```
using std::cout;
using std::endl;
#define PRESS_KEY std::cout<<"\nPresione Enter para continuar . . .\n";std::cin.get();

int main(int argc, char *argv[])
{
      Punto p(3000.12,4.45);
      Punto r;
      //  Punto q(5.7); -- Error de sintaxis
      cout <<"1. punto p: ("<<p.getX()<<";"<<p.getY()<<")"<<endl;
```

```
	cout <<"2. punto r: ("<<r.getX()<<";"<<r.getY()<<")"<<endl;
	r.setX(-2000.22);
	r.setY(3.33);
	cout <<"3. punto r: ("<<r.getX()<<";"<<r.getY()<<")"<<endl;
	p.setPunto(9900.9,8800.8);
	cout <<"4. punto p: ("<<p.getX()<<";"<<p.getY()<<")"<<endl;
	r=p.getPunto();
	cout <<"5. punto r: ("<<r.getX()<<";"<<r.getY()<<")"<<endl;
	PRESS_KEY;
}
```

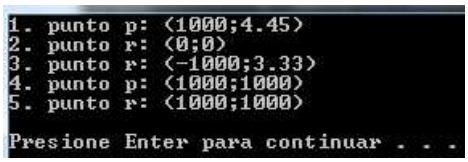
Salida:



Ejercicio 10.2

Modificar la clase del punto anterior, de manera de verificar que los valores de x;y se encuentren siempre dentro del rango -1000 a 1000, caso contrario su valor debe limitarse a estos valores.
De esta manera, la salida del código anterior sería:

Salida:



Nótese que los valores de x e y fueron limitados.
Este control se debe realizar a través de miembros privados de la clase.

Ejercicio 10.3

Modificar el punto anterior de manera que acepte también el Método **set_punto (pa)**; en donde **pa** es un objeto Punto y acepte la creación del objeto de manera que la expresión **Punto pc (34.4);** sea aceptada asignando el valor pasado como parámetro a la coordenada **x** y asigne el valor cero a la coordenada **y**.

Código Fuente:

```
int main(int argc, char *argv[])
{
	Punto p(1234.56);
	Punto r(12,34);
	cout <<"1. punto p: ("<<p.getX()<<";"<<p.getY()<<")"<<endl;
	cout <<"2. punto r: ("<<r.getX()<<";"<<r.getY()<<")"<<endl;
	p.setY(3.33);
	r.setPunto(p);
	cout <<"3. punto r: ("<<r.getX()<<";"<<r.getY()<<")"<<endl;
	PRESS_KEY;
}
```

Salida:

Ejercicio 10.4

Modificar el punto anterior de manera que acepte la creación de un objeto de la siguiente manera:
Punto pd (pc); o Punto pd=pc; // donde pc es un objeto Punto.

Código Fuente:

```
int main(int argc, char *argv[])
{
	Punto p(12.34,-56.78);
	Punto r(p);
	Punto q=p;
	cout <<"1. punto p: ("<<p.getX()<<";"<<p.getY()<<")"<<endl;
	cout <<"2. punto r: ("<<r.getX()<<";"<<r.getY()<<")"<<endl;
	cout <<"3. punto q: ("<<q.getX()<<";"<<q.getY()<<")"<<endl;
	PRESS_KEY;
}
```

Salida:

```
1. punto p: (12.34;-56.78)
2. punto r: (12.34;-56.78)
3. punto q: (12.34;-56.78)

Presione Enter para continuar . . .
```

¿Cuáles fueron las modificaciones necesarias? Justificar.

Ejercicio 10.5

Modificar el punto anterior para permitir la suma (+), resta (-) y asignación (=) de objetos tipo Punto.
Nota: tener en cuenta que las operaciones de deben controlar que los valores de x e y, no desborden el rango de +/- 1000.
No utilizar **friend**, para realizar la sobrecarga de los operadores suma (+) y resta (-).

Código Fuente:

```
int main(int argc, char *argv[])
{
	Punto p(12.34,-56.78);
	Punto r(1,4);
	Punto q;
	cout <<"1. punto p= ("<<p.getX()<<";"<<p.getY()<<")"<<endl;
	cout <<"2. punto r= ("<<r.getX()<<";"<<r.getY()<<")"<<endl;
	q=p+r;
	cout <<"3. punto p+r: q= ("<<q.getX()<<";"<<q.getY()<<")"<<endl;
	q=p-r;
	cout <<"4. punto p-r: q= ("<<q.getX()<<";"<<q.getY()<<")"<<endl;
	Punto s(990,-990);
	cout <<"5. punto s= ("<<s.getX()<<";"<<s.getY()<<")"<<endl;
	q=s+p;
	cout <<"6. punto s+p: q= ("<<q.getX()<<";"<<q.getY()<<")"<<endl;
	q=r+47;
	cout <<"7. punto r+47: q= ("<<q.getX()<<";"<<q.getY()<<")"<<endl;
	PRESS_KEY;
}
```

Salida:

```
1. punto p= (12.34;-56.78)
2. punto r= (1;4)
3. punto p+r: q= (13.34;-52.78)
4. punto p-r: q= (11.34;-60.78)
5. punto s= (990;-990)
6. punto s+p: q= (1000;-1000)
7. punto r+47: q= (48;4)

Presione Enter para continuar . . .
```

Preguntas:

- ¿Requiere sobrecargar el operador de asignación (=)? ¿Por qué?

- Justifique la ejecución de la línea `q=r+47;` (como se produce la suma entre el objeto Punto y el objeto int, suponiendo que mantuvo la consigna de escribir lo menos posible y no sobrecargó la suma de Punto con int)

Ejercicio 10.6

Modificar el punto anterior para permitir la ejecución del siguiente código. Recordar realizar siempre la menor cantidad de código. En este caso, sería que la implementación de la suma y resta se realice cada una con un solo método miembro.

Código Fuente:

```
int main(int argc, char *argv[])
{
	Punto p(12.34,-56.78);
	Punto r,s;
	s=78+p;
	r=78-p;
	cout <<"1. punto p= ("<<p.getX()<<";"<<p.getY()<<")"<<endl;
	cout <<"2. punto 78+p: s= ("<<s.getX()<<";"<<s.getY()<<")"<<endl;
	cout <<"3. punto 78-p: r= ("<<r.getX()<<";"<<r.getY()<<")"<<endl;
	r=p+s-45;
	cout <<"4. punto p+s-45: r= ("<<r.getX()<<";"<<r.getY()<<")"<<endl;
	PRESS_KEY;
}
```

Salida:

```
1. punto p= (12.34;-56.78)
2. punto 78+p: s= (90.34;-56.78)
3. punto 78-p: r= (65.66;56.78)
4. punto p+s-45: r= (57.68;-113.56)

Presione Enter para continuar . . .
```

Justificar la implementación.

Ejercicio 10.7

Modificar el punto anterior para permitir la comparación entre objetos tipo Punto: igualdad (==); desigualdad (!=), mayor (>) y menor (<), sobrecargar el flujo de salida y entrada (<< y >>).

En el caso de la sobrecarga de mayor (>) y menor (<), se debe comparar la distancia que existe al centro de coordenadas (0;0).

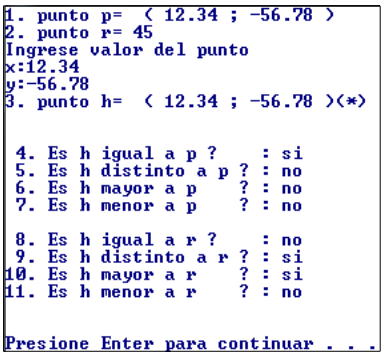
Código Fuente:

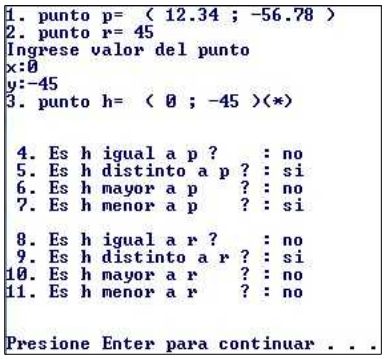
```
int main(int argc, char *argv[])
{
	Punto p(12.34,-56.78);
	double r=45;
	cout <<"1. punto p= "<<p<<endl;
	cout <<"2. punto r= "<<r<<endl;

	cout <<"Ingrese valor del punto"<<endl;
	Punto h;
	cin >>h;
	cout <<"3. punto h= "<<h<< "(*)" <<endl<<endl<<endl;
```

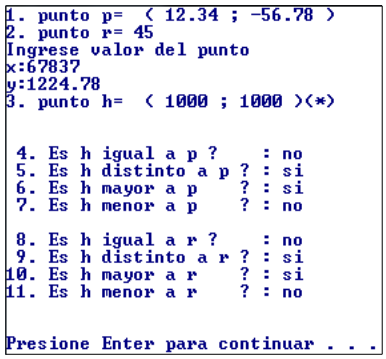
```
      cout <<" 4. Es h igual a p ?    : "<<((h==p)?"si":"no")<< endl;
      cout <<" 5. Es h distinto a p ? : "<<((h!=p)?"si":"no")<< endl;
      cout <<" 6. Es h mayor a p    ? : "<<((h>p)?"si":"no") << endl;
      cout <<" 7. Es h menor a p    ? : "<<((h<p)?"si":"no") << endl<<endl;

      cout <<" 8. Es h igual a r ?    : "<<((h==r)?"si":"no")<< endl;
      cout <<" 9. Es h distinto a r ? : "<<((h!=r)?"si":"no")<< endl;
      cout <<"10. Es h mayor a r    ? : "<<((h>r)?"si":"no") << endl;
      cout <<"11. Es h menor a r    ? : "<<((h<r)?"si":"no") << endl<<endl;
      PRESS_KEY;
}
```

(*) La salida luego del punto 3, dependerá del valor ingresado. Se muestran tres posibles ingreso con sus respectivas salidas. Nótese que si los valores están fuera del rango definido, estos valores son limitados.

Salidas:







Ejercicio 10.8

Agregar a la clase Punto, del ejercicio anterior, dos métodos públicos del tipo inline que indiquen la cantidad de instancias del tipo objeto que han sido creadas y cuantas se encuentran en memoria.
Nota: los contadores debe ser miembros privados de la clase.
Justificar la invocación de los métodos **getCantCreada** y **getCantExistente** antes de la creación de cualquier objeto.

Código Fuente:

```
void ff (void)
{
      Punto p,q,w;
      Punto h(34);
      Punto r=h;
      cout <<"a. Puntos Creados:"<<Punto::getCantCreada()<< " - Existentes:"<< r.getCantExistente()<<endl;
}

int main(int argc, char *argv[])
{
      cout <<"1. Puntos Creados:"<<Punto::getCantCreada()<< " - Existentes:"<< Punto::getCantExistente()<<endl;
      Punto p(12.34,-56.78);
      cout <<"2. Puntos Creados:"<<p.getCantCreada()<< " - Existentes:"<< p.getCantExistente()<<endl;
      Punto h(p);
      cout <<"3. Puntos Creados:"<<Punto::getCantCreada()<< " - Existentes:"<< Punto::getCantExistente()<<endl;
      ff();
      cout <<"4. Puntos Creados:"<<Punto::getCantCreada()<< " - Existentes:"<< Punto::getCantExistente()<<endl;
      PRESS_KEY;
}
```

Salida:

```
1. Puntos Creados:0 - Existentes:0
2. Puntos Creados:1 - Existentes:1
3. Puntos Creados:2 - Existentes:2
a. Puntos Creados:7 - Existentes:7
4. Puntos Creados:7 - Existentes:2

Presione Enter para continuar . . .
```

Ejercicio 10.9

Agregar un método público "**set_limites (float,float)**", que modifique el rango válido de x e y de la clase. Esto significa que el rango de todos los objetos existentes y de los objetos por crear se verán afectados por este método.
Este método no debe modificar los valores de coordenadas x;y. (No importa que éstos queden fuera de rango).
El primer parámetro corresponde al límite inferior y el segundo al superior. Si el límite inferior no es menor al superior, se debe omitir el cambio del rango.
Implementar además las funciones "**getLimiteSup**" y "**getLimiteInf**", del tipo **inline**, para saber cuáles son los valores de estos límites. Por omisión, los límites son +/- 1000.

Ejemplos:

`Punto Pa(5000,-5000);` toma el valor (1000;-1000) por los límites definidos por omisión.

Si luego ejecuto:

```
Punto::set_limites (50,-50);
Punto Pb(5000,-5000);
```

toma el valor (50;-50) por los límites definidos explícitamente.

Pa mantendrá el valor (1000;-1000), sin embargo si ejecuto `Pa=Pa+100`, su nuevo valor será (50;-50) pues se ve afectado por el nuevo rango definido.

Código Fuente:

```
int main(int argc, char *argv[])
{
	cout <<"1. Rango de punto: "<<Punto::getLimiteInf()<< ":"<<Punto::getLimiteSup()<<endl;

	Punto p(3000.12,5000);
	Punto r(12.34,34.56);
	cout <<"2. punto p: "<<p<<endl;
	cout <<"3. punto r: "<<r<<endl;

	Punto::setLimites(50,85);
	cout <<"4. Rango de punto: "<<p.getLimiteInf()<< ":"<<p.getLimiteSup()<<endl;
	cout <<"5. punto p: "<<p<<endl;
	cout <<"6. punto r: "<<r<<endl;

	Punto t;
	cout <<"7. nuevo punto t: "<<t<<endl;

	r=p;  // como la igualdad no esta redefinida, no se ve afectada por el nuevo límite
	cout <<"8.  r=p  r: "<<r<<endl;

	r.setPunto(p);
	cout <<"9. setPunto r: "<<r<<endl;

	r.setLimites(500,-85);
	cout <<"10. Rango de punto: "<<Punto::getLimiteInf()<< ":"<<Punto::getLimiteSup()<<endl;
	PRESS_KEY;
}
```

Nótese que al crear el objeto **t**, con parámetros por omisión, debería haber tomado el valor (0;0), sin embargo como el límite inferior es '50', toma el valor (50;50);
Además, como el operador igualdad (=) no está sobrecargado, la operación **r=p**, será una copia bit a bit y no se controlará el rango, por ende **r** poseerá valores fuera del rango.

Salida:

```
1. Rango de punto: -1000:1000
2. punto p:  ( 1000 ; 1000 )
3. punto r:  ( 12.34 ; 34.56 )
4. Rango de punto: 50:85
5. punto p:  ( 1000 ; 1000 )
6. punto r:  ( 12.34 ; 34.56 )
7. nuevo punto t:  ( 50 ; 50 )
8.  r=p  r:  ( 1000 ; 1000 )
9. setPunto r:  ( 85 ; 85 )
10. Rango de punto: 50:85

Presione Enter para continuar . . .
```

Pregunta: ¿por qué no se pude definir las funciones getLimiteSup y getLimiteInf como **const**?

Ejercicio 10.10

Sobrecargar el pre y post incremento de manera que incremente en una unidad tanto el valor de x como de y.

```
int main(int argc, char *argv[])
{
      Punto r(12.34,34.56);
      cout <<"1. punto r: "<<r<<endl;
      cout <<"2. punto r++: "<<r++<<endl;
      cout <<"3. punto r: "<<r<<endl;
      cout <<"4. punto ++r: "<<++r<<endl;
      PRESS_KEY;
}
```

Salida:

```
1. punto r:  ( 12.34 ; 34.56 )
2. punto r++:  ( 12.34 ; 34.56 )
3. punto r:  ( 13.34 ; 35.56 )
4. punto ++r:  ( 14.34 ; 36.56 )

Presione Enter para continuar . . .
```

Pregunta: que inconvenientes encuentra en una operación como la que sigue (revise los resultados obtenidos):

```
Punto x(10,10);
Punto z;
        z=x++ + x++;

int h=10;
int j;
        j=h++ + h++;
```

Ejercicio 10.11

Modificar el punto anterior de manera que acepte los operadores new y delete.

Código Fuente:

```
int main(int argc, char *argv[])
{
	Punto *r=new Punto(12.34,34.56);
	cout <<"1. punto r: "<<*r<<endl;
	cout <<"2. Puntos Creados:"<<r->getCantCreada()<< " - Existentes:"<< Punto::getCantExistente()<<endl;
	delete (r);
	cout <<"3. Puntos Creados:"<<Punto::getCantCreada()<< " - Existentes:"<< Punto::getCantExistente()<<endl;
	PRESS_KEY;
}
```

Salida:

```
1. punto r:  ( 12.34 ; 34.56 )
2. Puntos Creados:1 - Existentes:1
3. Puntos Creados:1 - Existentes:0

Presione Enter para continuar . . .
```

¿Cuáles fueron las modificaciones necesarias? Justificar.

Ejercicio 10.12

Se debe implementar la clase IntArr, la cual es una clase para trabajar con una array o vector de datos tipo **int** en forma dinámica, cuya definición es la que sigue:

```
class IntArr
{
private:
	int * p;
	int size;
	int used;
 public:
	IntArr (int sz);
	IntArr (int sz,int qtty,int *vec);
	~IntArr();
	void prtArr (void) const;
};
```

En donde

p: es el puntero al array dinámico.
size: es la cantidad de elementos del array **p**.
used: es la cantidad de elementos usados del array **p**. **used** será siempre <= a **size**.

IntArr (int sz): crea un elemento **IntArr** con **sz** elementos disponibles.
IntArr (int sz,int qtty, int *vec): crea un elemento **IntArr** con **sz** elementos disponibles, y copia **qtty** elementos del array **vec** al array **p**.
~**IntArr (int sz)**: destructor.

prtArr: imprime el Array en pantalla.

Como ejemplo de aplicación se presenta el siguiente ejemplo con su respectiva salida:

Nota: si en IntArr (sz,qtty,vec); **qtty** es mayor a **sz**, se debe solucionar el error igualando **sz** a **qtty**.

Código Fuente:

```
int main(int argc, char *argv[])
{
	IntArr A(30);
	int v_aux[]={23,4,54,634,6677,32,56};
	IntArr B(40,sizeof(v_aux)/sizeof(int),v_aux);
	A.prtArr();
	B.prtArr();
	PRESS_KEY;
}
```

Salida:

```
> Array: Vacio !!!
> Array:23 ; 4 ; 54 ; 634 ; 6677 ; 32 ; 56

Presione Enter para continuar . . .
```

Ejercicio 10.13

Agregar a la clase IntArr, los siguientes métodos públicos:

getSize del tipo inline que devuelve el tamaño del IntArr.
getUsed del tipo inline que devuelve la cantidad de elementos usados del IntArr.

getAvg devuelve el promedio de los elementos del IntArr. El promedio devuelto es del tipo double.

Y sobrecargar el método prtArr, de manera de indicar cuantos elementos imprimir desde el inicio del Array.

Código Fuente:

```
int main(int argc, char *argv[])
{
	IntArr A(30);
	int v_aux[]={23,4,54,634,6677,32,56};
	IntArr B(40,sizeof(v_aux)/sizeof(int),v_aux);
	A.prtArr();
	B.prtArr();
	B.prtArr(3);
	cout<<"\n\nObjeto B -"<<endl;
	cout<<" size:"<<B.getSize()<<endl;
	cout<<" used:"<<B.getUsed()<<endl;
	cout<<" promedio:"<<B.getAvg()<<endl;
	PRESS_KEY;
}
```

Salida:

```
> Array: Vacio !!!
> Array:23 ; 4 ; 54 ; 634 ; 6677 ; 32 ; 56
> Array:23 ; 4 ; 54


Objeto B -
 size:40
 used:7
 promedio:1068.57

Presione Enter para continuar . . .
```

Ejercicio 10.14

Agregar a la clase IntArr, los siguientes métodos públicos:

addElement (int xx): el cual agrega el elemento **xx** al final del array "IntArr".
addElement (**int qtty, int *vec)**: el cual agrega los **qtty** elementos que se encuentran en **vec** al final del array "IntArr".

Nota: tener en cuenta que al agregar elementos al objeto, se puede desbordar el Array dinámico, por lo tanto habrá que redimensionarlo. En tal caso y en forma preventiva solicitar espacio para 5 elementos más de los necesarios.

Código Fuente:

```
int main(int argc, char *argv[])
{
	int v_aux[]={0,5,10,15,20,25,30,35,40};
	IntArr A(10,sizeof(v_aux)/sizeof(int),v_aux);
	cout<<" size:"<<A.getSize()<<endl<<" used:"<<A.getUsed()<<endl;
	A.prtArr();
	A.addElement(77);
	cout<<" size:"<<A.getSize()<<endl<<" used:"<<A.getUsed()<<endl;
	A.prtArr();
	A.addElement(11);
	cout<<" size:"<<A.getSize()<<endl<<" used:"<<A.getUsed()<<endl;
	A.prtArr();
	A.addElement(8,v_aux);
	cout<<" size:"<<A.getSize()<<endl<<" used:"<<A.getUsed()<<endl;
	A.prtArr();
	PRESS_KEY;
}
```

Salida:

```
 size:10
 used:9
> Array:0 ; 5 ; 10 ; 15 ; 20 ; 25 ; 30 ; 35 ; 40
 size:10
 used:10
> Array:0 ; 5 ; 10 ; 15 ; 20 ; 25 ; 30 ; 35 ; 40 ; 77
 size:16
 used:11
> Array:0 ; 5 ; 10 ; 15 ; 20 ; 25 ; 30 ; 35 ; 40 ; 77 ; 11
 size:24
 used:19
> Array:0 ; 5 ; 10 ; 15 ; 20 ; 25 ; 30 ; 35 ; 40 ; 77 ; 11 ; 0 ; 5 ; 10 ; 15 ; 2
0 ; 25 ; 30 ; 35

Presione Enter para continuar . . .
```

Ejercicio 10.15

Modificar los métodos **addElement**, incluidos en el ejercicio anterior, de manera de poder indicar a partir de qué se deben agregar los nuevos elementos. Siendo el nuevo prototipo como sigue:

addElement (**int pos**, int xx).
addElement (**int pos**, int qtty, int *vec)

En donde pos, indica la posición a partir de donde comienza la inserción. Los demás parámetros mantienen su significado.

Nota: En caso que **pos** fuese negativo, se considera inserción desde el inicio y sí el valor de pos, supera la máxima ubicación posible dentro del Array dinámico, se debe insertar al final.

Código Fuente:

```
#define SZ_VEC(x) (sizeof(x)/sizeof(x[0]))
int main(int argc, char *argv[])
{
      int v1[]={0,5,10,15,20,25,30,35,40};
      int v2[]={1,2,3,4,5,6};
      IntArr A(10,SZ_VEC(v1),v1);
      cout<<" size:"<<A.getSize()<<endl<<" used:"<<A.getUsed()<<endl;
      A.prtArr();
      A.addElement(0,77);
      A.addElement(56,11);
      A.addElement(4,SZ_VEC(v2),v2);
      cout<<" size:"<<A.getSize()<<endl<<" used:"<<A.getUsed()<<endl;
      A.prtArr();
      A.addElement(4,99);
      cout<<" size:"<<A.getSize()<<endl<<" used:"<<A.getUsed()<<endl;
      A.prtArr();
      PRESS_KEY;
}
```

Salida:

```
 size:10
 used:9
> Array:0 ; 5 ; 10 ; 15 ; 20 ; 25 ; 30 ; 35 ; 40
 size:22
 used:17
> Array:77 ; 0 ; 5 ; 10 ; 1 ; 2 ; 3 ; 4 ; 5 ; 6 ; 15 ; 20 ; 25 ; 30 ; 35 ; 40 ;
11
 size:22
 used:18
> Array:77 ; 0 ; 5 ; 10 ; 99 ; 1 ; 2 ; 3 ; 4 ; 5 ; 6 ; 15 ; 20 ; 25 ; 30 ; 35 ;
40 ; 11

Presione Enter para continuar . . .
```

Ejercicio 10.16

Agregar a la clase IntArr del ejercicio anterior, los elementos necesarios para permitir las sentencias que se visualizan en el siguiente código:

Código Fuente:

```
#define SZ_VEC(x) (sizeof(x)/sizeof(x[0]))
int main(int argc, char *argv[])
{
	cout<<"\n==================> Inicio <==================\n";
	int v1[]={0,5,10,15,20,25,30,35,40};
	int v2[]={1,2,3,4,5,6};
	IntArr A(10,SZ_VEC(v1),v1);
	IntArr B(10,SZ_VEC(v2),v2);
	IntArr C=B;
	B.addElement(0,99);
	cout<<" size:"<<A.getSize()<<endl<<" used:"<<A.getUsed()<<endl;
	A.prtArr();
	cout<<"\n> Array B\n"<<B;
	cout<<"\n> Array C\n"<<C;
	cout<<"\n==================> Medio <==================\n";
	A=B+C;
	cout<<"\n> Array A=B+C\n"<<A;
	IntArr D(10,SZ_VEC(v1),v1);
	D=D;
	cout<<"\n> Array A\n"<<A;
	cout<<"\n==================> Medio <==================\n";
	D+=B;
	cout<<"\n> Array D+=B\n"<<D;
	PRESS_KEY;
}
```

Salida:

```
==================> Inicio <==================
 size:10
 used:9
> Array:0 ; 5 ; 10 ; 15 ; 20 ; 25 ; 30 ; 35 ; 40

> Array B
Array (size:10)-(used:7)
> Array:99 ; 1 ; 2 ; 3 ; 4 ; 5 ; 6

> Array C
Array (size:10)-(used:6)
> Array:1 ; 2 ; 3 ; 4 ; 5 ; 6

==================> Medio <==================

> Array A=B+C
Array (size:18)-(used:13)
> Array:99 ; 1 ; 2 ; 3 ; 4 ; 5 ; 6 ; 1 ; 2 ; 3 ; 4 ; 5 ; 6

> Array A
Array (size:18)-(used:13)
> Array:99 ; 1 ; 2 ; 3 ; 4 ; 5 ; 6 ; 1 ; 2 ; 3 ; 4 ; 5 ; 6

==================> Medio <==================

> Array D+=B
Array (size:21)-(used:16)
> Array:0 ; 5 ; 10 ; 15 ; 20 ; 25 ; 30 ; 35 ; 40 ; 99 ; 1 ; 2 ; 3 ; 4 ; 5 ; 6

Presione Enter para continuar . . .
```