Constructores: Ctors

Anteriormente se han utilizado dos tipos de constructores, en el primer tipo se enviaba algún parámetro a este, (la clase figura recibía en su constructor el nombre de la figura), en el segundo caso no se enviaba ningún parámetro a la clase (esto pasaba con las clases triángulo, circulo y cuadrado, sus constructores no recibían ningún parámetro, lo que hacían era fijar el nombre de la clase figura).

Como cualquier otra función en C++, los constructores se pueden sobrecargar. La sobrecarga de funciones es un poderoso método, el cual permite dar el mismo nombre a funciones diferentes, lo único que debe cambiar es el tipo de los parámetros de entrada. Por ejemplo:

```
int cuadrado(int x){
  return(x*x);
}
float cuadrado (float x){
  return(x*x);
}
```

En el ejemplo anterior, se tienen dos funciones diferentes las cuales poseen el mismo de nombre; de esta forma al utilizarlas se tienen lo siguiente:

```
int main(void){
  int i=10, ri;
  float k=2.5, rf;
  ri = cuadrado(i);
  rf = cuadrado(k);
}
```

Para el usuario de clases se tiene una única función “cuadrado” la cual, sin saber como, es capaz de elevar al cuadrado cualquier tipo de datos que le den, esto es sobrecarga de funciones. Ahora el trabajo de unir las funciones respectivas le toca al compilador, este cuando se llama una función ve su nombre y los parámetros que se le envían a la función, y busca cual función de todas las existentes cumple con estas dos características, un error existirá tanto si hay más de una iguales como si no hay ninguna.

Volviendo a los constructores, es posible tener varias versiones de constructores, sólo se cambia el tipo de los parámetros de entrada, algunas veces se desea también tener la posibilidad de que un MISMO constructor no reciba parámetros o los reciba, es decir tener un constructor por defecto. Esto se puede hacer poniendo los parámetros por defecto en el prototipo del constructor, por ejemplo en el constructor de la clase figura, se tenía que este asignaba un nombre a la figura a crear, pero que pasa si se desea tener la posibilidad de no enviar un nombre al objeto figura a crear. Esto se puede resolver de dos formas, la primera y menos elegante es declarar otro constructor pero definiendo su lista de parámetros como void, la otra es creando un constructor por defecto, esto se hace cambiando la declaración del prototipo del constructor en el archivo figura.h por algo como esto:

```
C_figura(string="figuraSinNombre");
```

La implementación del constructor quedaría igual, esto hace que si al crear un objeto tipo figura se da un nombre este se asignara, si no se da un nombre, el nombre por defecto figuraSinNombre, será asignado.

En los ejemplos anteriores no se había tratado, pero existe lo que se llama un constructor de copia, el prototipo de este constructor tienen la forma:

```
C_mi_calse(const C_mi_clase &);
```

Este constructor recibe un elemento del tipo de la clase al cual corresponde, y su función es la de hacer una copia del objeto que se pasa como parámetro al constructor del objeto que se está creando. La función de copia debe hacer no existe por defecto.

Memoria dinámica

En el ejemplo de la pila, se asignaba memoria en forma dinámica, es decir cada vez que se apilaba un nuevo dato, se pedía memoria para él. En ese ejemplo la memoria se asignaba un dato a la vez, pero que pasa si en lugar asignar memoria un dato a la vez se desea asignar memoria para todo un arreglo, la sintaxis para hacer esto es:

```
ptr_data = new int[N]
```

para direccionar un dato de este arreglo se hace:

```
ptr_data[n];
```

Donde $n \in [0, N]$.

Para recuperar la memoria asignada se hace:

```
delete [] ptr_data;
```

Clase Matriz: Version 1

En este laboratorio se elaborará una clase matriz, capaz de sumar matrices, multiplicarlas por un escalar y otras operaciones más. Lo primero que se hará es crear una matriz básica, con el siguiente prototipo:

```
//-------------------------------------------------------------------------/
//matriz1.h
//-------------------------------------------------------------------------/

#ifndef MATRIZ_H
#define MATRIZ_H
/*************************************************************************/
class C_matriz{
public:
  C_matriz(int=1, int=1);
  //par ámetros por defecto, estos son fila, columna
  C_matriz(const C_matriz &);
  //constructor de copia
  ~C_matriz();
  float get(int fila, int columna) const; //retorna un elemento de la matriz
```

```
  //graba un elemento en una óposicin de la matriz
  void set(int fila, int columna ,float dato);
private:
  int filas;
  int columnas;
  float *ptr_data;
};
/*************************************************************************/
#endif
```

El programa principal de prueba sería:

```
//-----------------------------------------------------------------------/
//principal1.cpp
//-----------------------------------------------------------------------/

#include "matriz1.h"
#include <iostream>
/*********************************************************/
void desplegar_matriz(int ,int ,const C_matriz &);
void llenar_matriz(int x,int y,C_matriz &);
/*********************************************************/
using namespace std;
/*********************************************************/
int main(void){
  const int n=2;
  const int m=3;
  cout<<"Probando el constructor y get():"<<endl;
  C_matriz A(n,m);
  desplegar_matriz(n,m,A); //use el miembro get()
  cout<<"Probando set():"<<endl;
  llenar_matriz(n,m,A); //use el miembro set()
  desplegar_matriz(n,m,A);
  cout<<"Probando el constructor con par ámetros predeterminados:"<<endl;
  C_matriz B;
  desplegar_matriz(1,1,B);
  cout<<"Probando el constructor de copia:"<<endl;
  C_matriz C(A);
  desplegar_matriz(n,m,C);
  return 0;
}
/*********************************************************/
void desplegar_matriz(int x,int y,const C_matriz &Matriz){
for(int i=1;i<=x;i++){
  for(int j=1;j<=y;j++){
    cout<<Matriz.get(i,j)<<"\t";
  }
  cout<<endl;
  //siguiente fila
  }
};
```

```
/*************************************************************/
void llenar_matriz(int x,int y,C_matriz &Matriz){
  float elemento=0;
  for(int i=1;i<=x;i++){
    for(int j=1;j<=y;j++){
      Matriz.set(i,j,elemento);
      elemento++;
    }
  }
};
/*************************************************************/
```

Clase Matriz: Version 2

A continuación se le agregarán más funcionalidades a la clase matriz, para ello se sobrecargarán algunos operadores.

El nuevo prototipo de la clase matriz es:

```
//-------------------------------------------------------------------------/
//matriz2.h
//-------------------------------------------------------------------------/

#ifndef MATRIZ_H
#define MATRIZ_H
/*************************************************************************/
class C_matriz{
public:
  C_matriz(int=1, int=1);
  //par ámetros por defecto, estos son fila, columna
  C_matriz(const C_matriz &);
  //constructor de copia
  ~matriz();
  float get(int fila, int columna) const; //retorna un elemento de la matriz
  //graba un elemento en una óposicin de la matriz
  void set(int fila, int columna ,float dato);
  C_matriz &operator=(const C_matriz &); //Operador de asignaci ón
  float operator()(int, int);
  //Operador para seleccionar un elemento en la matriz, similar a get
  void desplegar(void);
  //desplegar toda la matriz, por filas y columnas
  matriz operator+(const C_matriz &);
  //suma dos matrices
  matriz operator*(const float &);
  //multiplica una matriz por un flotante
private:
  int filas;
  int columnas;
  float *ptr_data
```

```
};
/*************************************************************************/
#endif
```

El programa principal correspondiente a esta clase es el siguiente:

```
//-----------------------------------------------------------------------/
//principal2.cpp
//-----------------------------------------------------------------------/

#include "matriz2.h"
#include <iostream>
/*********************************************************/
void llenar_matriz(int x,int y,C_matriz &);
/*********************************************************/
using namespace std;
/*********************************************************/
int main(void){
  const int n=2;
  const int m=3;
  cout<<"Probando los operadores sobrecargados =, + and *:"<<endl;
  cout<<"A:"<<endl;
  C_matriz A(n,m);
  llenar_matriz(n,m,A);
  A.desplegar();
  cout<<"B:"<<endl;
  C_matriz B(A);
  B=A*2;
  B.desplegar();
  cout<<"C:"<<endl;
  C_matriz C=A+B;
  C.desplegar();
  cout<<C"(1,1)<<\n "<<C(2,2)<< endl;
  return 0;
}
/*********************************************************/
void llenar_matriz(int x,int y,C_matriz &Matriz){
  float element=0;
  for(int i=1;i<=x;i++){
    for(int j=1;j<=y;j++){
      Matriz.set(i,j,element);
      element++;
    }
  }
};
/*********************************************************/
```

Clase Matriz: Version 3

En la versión anterior de la clase matriz, se sobrecargo el operador de multiplicación para poder realizar el pro-

ducto entre una matriz y un escalar, al usar este operador se hacían cosas como resultado = MiMatriz * 3.1415; pero que sucede si se de sea hacer algo como resultado = 3.1415 * MiMatriz; esto debería servir, técnicamente la multiplicación es conmutativa, pero si usted escribe la línea anterior tendrá un error, este es, no existe una función llamada '*' que recibe un flotante y una matriz, sólo hay una función llamada '*' que recibe una matriz y un flotante.

Lo anterior sucede porque al sobrecargar un operador como miembro de una clase, el primer operando debe ser del tipo de la clase que se está definiendo. Lo anterior se soluciona declarando el operador fuera de la clase, y haciendo esta función amiga de la clase.

Una función amiga es una función cualquiera, pero tiene cierta particularidad, esta es, una función amiga puede acceder los miembros y métodos protegidos y privados de la clase. Como usted recordará lo que está bajo las premisas de privado y protegido no puede ser accedido por entes externos a la clase; la forma de permitirle a un ente externo violar el principio de ocultamiento es declararla amiga de la clase.

La forma de hacer una función amiga, es poner un prototipo de la función en la declaración de la clase, y anteponerle a este prototipo la directiva friend.

A continuación se da la declaración de la nueva versión de matriz:

```
//-------------------------------------------------------------------------/
//matriz3.h
//-------------------------------------------------------------------------/

#ifndef MATRIZ_H
#define MATRIZ_H
/*************************************************************/
#include <iostream>
/*************************************************************/
using namespace std;
/*************************************************************/
class C_matriz{
  friend ostream &operator<<(ostream &,const C_matriz &); //Similar a desplegar()
  friend C_matriz operator*(const C_matriz &,const float &);
  //multiplica una matriz por un flotante
  friend C_matriz operator*(const float &,const C_matriz &);
  // multiplica un flotante por una matriz
  public:
  C_matriz(int=1, int=1);
  //par ámetros por defecto, estos son fila, columna
  C_matriz(const C_matriz &);
  //constructor de copia
  ~C_matriz();
  float get(int fila, int columna) const; //retorna un elemento de la matriz
  //graba un elemento en una óposicin de la matriz
  void set(int fila, int columna ,float dato);
  C_matriz &operator=(const C_matriz &); //Operador de asignaci ón
  float operator()(int, int);
  //Operador para seleccionar un elemento en la matriz, similar a get
  void desplegar(void);
  //desplegar toda la matriz, por filas y columnas
```

```
  C_matriz operator+(const C_matriz &);
  //suma dos matrices
  C_matriz operator-(const C_matriz &);
  //resta dos matrices
  private:
  int fila;
  int columna;
  float *ptr_data
};
/*************************************************************/
#endif
```

La función principal para esta clase es:

```
//-------------------------------------------------------------------------/
//principal3.cpp
//-------------------------------------------------------------------------/

#include "matriz3.h"
#include <iostream>
/*************************************************************/
void llenar_matriz(int x,int y,C_matriz &);
/*************************************************************/
using namespace std;
/*************************************************************/
int main(){
  const int n=2;
  const int m=3;
  cout<<" Probando los operadores sobrecargados =, + and *:"<<endl;
  cout<<"A:"<<endl;
  C_matriz A(n,m);
  llenar_matriz(n,m,A);
  cout<<A<<endl;;
  A.desplegar();
  cout<<"B:"<<endl;
  C_matriz B(A);
  B=3*A;
  cout<<B<<endl;;
  B.display();
  cout<<"C:"<<endl;
  C_matriz C=B-A;
  cout<<C<<endl;
  C.desplegar();
  return 0;
}
/*************************************************************/
void llenar_matriz(int x,int y,C_matriz &Matriz){
  float elemento=0;
  for(int i=1;i<=x;i++){
    for(int j=1;j<=y;j++){
      Matriz.set_element(i,j,elemento);
```

```
      elemento++;
    }
  }
};
/*************************************************************/
```

Dado que la sintaxis para la función amiga de desplegar no es muy intuitiva esta función se da aquí, inclúyala en el archivo matriz3.cpp

```
/*************************************************************/
ostream & operator<<(ostream & Output,const C_matriz &Matriz){
  for(int i=1;i<=Matriz.filass;i++){
    for(int j=1;j<=Matriz.columnas;j++){
      Output<<Matriz(i,j)<<"\t"; //Output puede ser algo como cout
    }
    cout<<endl;
  }
  return Output;
};
/*************************************************************/
```