Universidad Autónoma del Estado de Hidalgo

Instituto de Ciencias Básicas e Ingeniería

Área Académica de Computación y Electrónica

Licenciatura en Ciencias Computacionales

Programación Orientada a Objetos

Docente: M. en C. Iliana Castillo Pérez

Tema:
Constructores en C++

Abstract:

In this document is shown the constructor description as a mechanism to ensure objects whit valid values or without invalid values.
Also, the general syntax, a declaration example and a practical example are presented for each of the five constructors that have been proposed by different authors.

Keywords: Constructors, Object initialization.

Tema:
Constructores en C++

Resumen :

En este documento se presenta la descripción del constructor como un medio para asegurar que los objetos inicien con valores válidos o sin basura. También se muestra la sintaxis general, un ejemplo de declaración y un ejemplo práctico de cada uno de los cinco constructores que han sido propuestos por diversos autores.

Palabras Clave: Constructores, Inicialización de objetos.

Tema:
Constructores en C++

Introducción:

En C++ una forma de asegurar que los objetos siempre contengan valores válidos es escribir un constructor.

Un constructor es una función miembro especial de una clase que es llamada automáticamente siempre que se declara un objeto de esa clase. Su función es crear e inicializar un objeto de su clase [1],[2],[3].

Definición:

En C++ la inicialización de los datos miembro del objeto no se puede realizar en el momento en que son declarados; sin embargo, existe una función miembro especial llamada constructor que permite inicializar objetos en el momento en que se crean.

Un constructor es una función miembro especial que se llama exactamente igual que la clase y sirve para construir un nuevo objeto y asignar valores válidos a sus datos miembro [1],[2],[3].

Características:

- Tiene el mismo nombre que la clase a la que pertenece.
- No tiene tipo de retorno (incluyendo void), por lo tanto no retorna ningún valor.
- Puede admitir parámetros como cualquier otra función.
- No puede ser heredado.
- No puede ser declarado *const* ni *static*.
- Debe ser público.

Creación de objetos:

Cuando una clase tiene un constructor, éste será invocado automáticamente siempre que se cree un nuevo objeto de esa clase [1].

Se puede crear un objeto de cualquiera de las formas siguientes:

- Declarando un objeto global
- Declarando un objeto local u objeto temporal
- Invocando al operador new
- Llamando explícitamente a un constructor

Tipos de constructores

Existen 5 tipos de constructores:

1) Constructor por omisión (implícito)
2) Constructor explícito con argumentos
3) Constructor explícito con argumentos por omisión
4) Constructor copia
5) Constructor explícito o por defecto

Constructor por omisión (implícito)

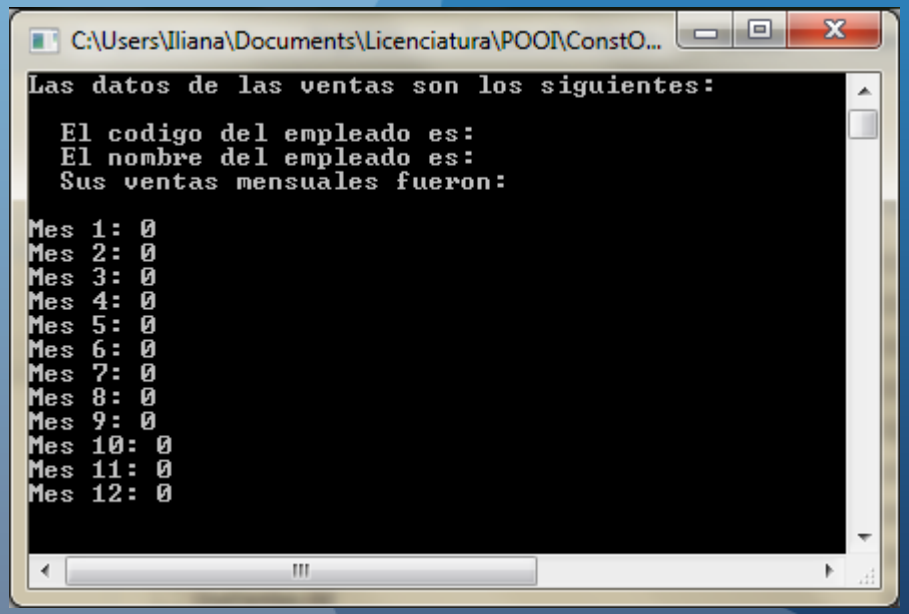
Dado que los constructores son funciones miembro, admiten argumentos igual que éstas. Cuando en una clase no especificamos ningún constructor el compilador crea uno por omisión sin argumentos [1].

Un constructor por omisión de una clase, es un constructor que se puede invocar sin argumentos.

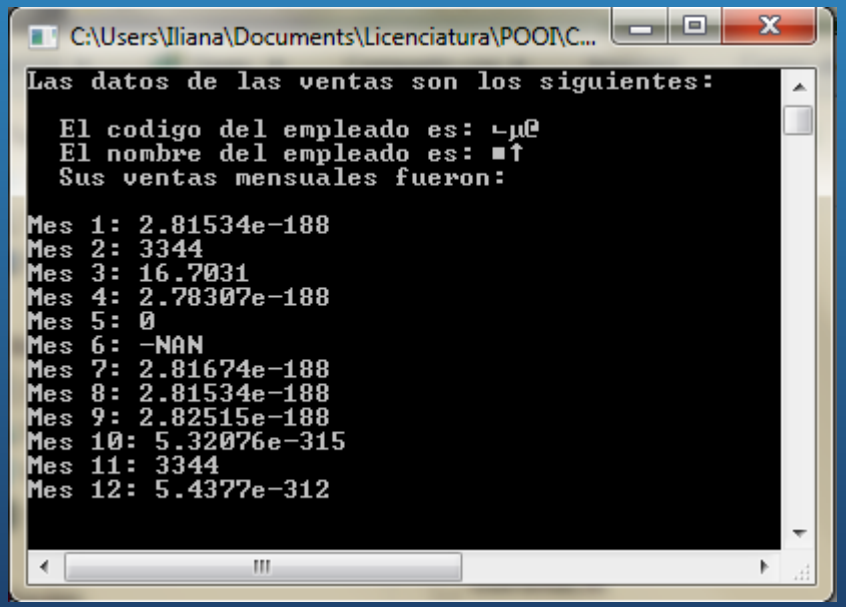
Importante:

Si el objeto creado con el constructor por omisión es **global**, el constructor inicializa a ceros los datos miembro numéricos y a nulos los datos miembro alfanuméricos. Si el objeto creado es **local**, lo inicializa con valores indeterminados (basura) [1].

Ejemplo:



Objeto Global



Objeto Local

Constructor explícito con argumentos

Un constructor explícito con argumentos es aquel que define una lista de argumentos que le ayudarán a inicializar el objeto con valores válidos, en dicha lista de argumentos recibe los valores con los cuales inicializará el objeto que se va a construir [1],[2],[3],[4],[5]. Su sintaxis es la siguiente:

Clase(lista de variables con tipo);

Ejemplo:

Persona(char nombre[30], char curp[20], int edad);

El ejemplo práctico que se muestra abajo fue tomado de [1].

```
#include <iostream.h>

class CFecha
{
    private:
            int dia, mes, anio;
    protected:
            int bisiesto();
    public:
            CFecha(int, int, int);    //constructor explícito con argumentos
            void asignarFecha( );
            void obtenerFecha( );
            int fechaCorrecta( );
};

CFecha::CFecha(int dd, int mm, int aa)
{
  dia = dd;
  mes = mm;
  anio = aa;
}
/*Continua el resto de las funciones miembro de la clase.*/
```

Cuerpo del constructor explícito con argumentos

Una vez declarado el constructor explícito con argumentos en la clase [1], la declaración de los objetos en el programa sería de la siguiente forma:

```
#include <iostream.h>
#include <conio.h>
#include "CFecha.h"  //Nombre del archivo de la clase CFecha

void main( )
{
  CFecha fecha1(13,3,2014);  //Ejecuta el constructor explícito con argumentos
  CFecha fecha2(23,8,2014);

  fecha1.obtenerFecha( );
  fecha2.obtenerFecha( );
  getch( );
}
```

Una vez que se ha declarado un constructor, como en el caso del constructor explícito con argumentos, éste substituye al constructor por omisión que genera el compilador y ya no podemos declarar objetos sin enviar argumentos como en el caso:

CFecha fecha3;

Esta línea generaría un error ya que el nuevo constructor requiere tres argumentos enteros. Para solucionar este problema, debemos agregar a la clase el constructor por omisión [1].

El ejemplo práctico que se muestra abajo fue tomado de [1].

```
#include <iostream.h>
class CFecha
{
  private:
            int dia, mes, anio;
  protected:
int Bisiesto( );
  public:
      CFecha( ){ };              // constructor por omisión
      CFecha(int, int, int);    //constructor explícito con argumentos
      void asignarFecha( );
      void obtenerFecha( );
      int fechaCorrecta( );
};

CFecha::CFecha(int dd, int mm, int aa)
{
  dia = dd;
  mes = mm;
  anio = aa;
}    /*Continua el resto de las funciones miembro de la clase.*/
```

Una vez declarado en la clase el constructor por omisión, será posible generar objetos en el programa, ya sea enviando los tres argumentos o sin enviar argumentos [1], tal como se muestra en el siguiente código:

```
#include <iostream.h>
#include <conio.h>
#include "CFecha.h"

void main( )
{
  CFecha fecha1(13,5,2014); //Objeto que utiliza el constructor explícito con
                                                    argumentos
  CFecha fecha2(23,11,2014);
  CFecha fecha3;   // objeto que llama automáticamente al constructor por omisión

  fecha1.obtenerFecha( );
  fecha2.obtenerFecha( );
  fecha3.obtenerFecha( );
  getch( );
}
```

Para el caso del constructor explícito con argumentos, C++ recomienda utilizar siempre que sea posible la siguiente sintaxis:

```
CFecha::CFecha(int dd, int mm, int aa): dia(dd), mes(mm), anio(aa)
{
}
```

Los dos puntos a continuación de la lista de parámetros del constructor CFecha indica que sigue una lista de inicializadores, en este caso, los datos miembro de la clase. Observe también que el cuerpo del constructor aparece vacío puesto que no se requiere ninguna operación adicional [1].

Constructor explícito con argumentos por omisión

Una forma de reducir el número de constructores, es utilizar constructores con argumentos por omisión, este tipo de constructor declara una lista de argumentos a continuación del nombre del constructor y además los inicializa con algún valor predefinido, permitiendo hacer llamadas al constructor sin argumentos o incluso con uno, dos, tres o n argumentos según sea el caso [1],[5]. Su sintaxis es la siguiente:

Clase(lista de variables con tipo inicializadas);

Ejemplo:

CFecha(int dd=1, int mm=1, int aa=1990);

El ejemplo práctico que se muestra abajo fue tomado de [1].

```
#include <iostream.h>
class CFecha
{
  private:
            int dia, mes, anio;
  protected:
            int Bisiesto( );
  public:
            CFecha(int dd=1, int mm=1, int aa=1990);    //c. explícito con arg. por omisión
            void asignarFecha( );
            void obtenerFecha( );
            int fechaCorrecta( );
};

CFecha::CFecha(int dd, int mm, int aa)
{
  dia = dd;
  mes = mm;
  anio = aa;
}
/*Continua el resto de las funciones miembro de la clase.*/
```

Cuerpo del constructor explícito con argumentos por omisión

Como se puede observar, el cuerpo del constructor es el mismo que en el caso del constructor explícito con argumentos, la única diferencia radica en la declaración del constructor, que es donde se inicializan los parámetros [1].

Ahora, al momento de declarar y construir los objetos en el programa, podremos hacerlo enviándole diferente número de argumentos, sin necesidad de tener otros constructores, ejemplo:

```
/* Cabeceras */
void main( )
{
  CFecha fecha;
  CFecha fecha1(13);
  CFecha fecha2(13,11);
  CFecha fecha3(13,11,2014);

  fecha.obtenerFecha( );
  fecha1.obtenerFecha( );
  fecha2.obtenerFecha( );
  fecha3.obtenerFecha( );
}
```

Se ejecuta el constructor explícito con argumentos por omisión

Constructor copia

Otra forma de inicializar un objeto es asignándole otro objeto de la misma clase en el momento de su declaración, esto es, cuando se crea [1]. Por ejemplo:

```
CFecha hoy(18,2,2014);
CFecha otroDia(hoy);
CFecha unDia=hoy;
```

En este ejemplo, se crea y se inicializa el objeto *hoy* y a continuación se crean los objetos *otroDia* y *unDia* inicializándolos con el objeto *hoy*, de tal forma que todos los objetos creados tienen como fecha inicial 18/2/2014.

Un constructor que crea un nuevo objeto a partir de otro existente es denominado constructor copia. Un constructor de éstas características tiene un solo argumento, el cual es una referencia a un objeto constante de la misma clase [1],[2],[3],[5]. Su sintaxis es la siguiente:

Clase(const Clase &obj);

Ejemplo:

Persona(const Persona &obPerson);

Donde:

Persona es el nombre de la clase.

const es palabra reservada que se emplea para designar constantes.

& es el operador para hacer la referencia.

obPerson es el nombre que le queremos dar al objeto.

Si no especificamos un constructor copia, el compilador proporciona un constructor copia por omisión público, para cada clase definida. También cuando sea necesario podemos codificar nuestra propia versión del constructor copia [1],[2],[3],[5].

NOTA: Es importante señalar que dentro de la clase, el constructor copia debe ir acompañado siempre por algún otro constructor, debido a que es necesaria la referencia a un objeto ya construido para poder realizar la copia.

El constructor copia [1], de la clase CFecha sería:

```
#include <iostream.h>
class CFecha
{
  private:
          int dia, mes, anio;
  protected:
          int Bisiesto( );
  public:
          CFecha(int=1, int=1, int=1990);   //constructor explicito con arg. por omisión
          CFecha(const CFecha &obFecha);    //constructor copia
          void asignarFecha( );
          void obtenerFecha( );
          int fechaCorrecta( );
};

CFecha::CFecha(const CFecha &obFecha)
{
  dia = obFecha.dia;
  mes = obFecha.mes;
  anio = obFecha.anio;
}
/*Continua el resto de las funciones miembro de la clase.*/
```

Cuerpo del constructor copia

Vemos que el constructor copia acepta como argumento una *referencia* al objeto a copiar y que el cuerpo del constructor copia, asigna miembro a miembro ese objeto en el nuevo objeto construido, motivo por el cual se llama constructor copia [1]. El programa sería el siguiente:

```
/* Cabeceras */
#include " CFecha.h"
void main( )
{
    CFecha fecha(13,11,2014);
    CFecha fecha1=fecha;
    CFecha fecha2(fecha);

  fecha.obtenerFecha( );
  fecha1.obtenerFecha( );
  fecha2.obtenerFecha( );
}
```

Constructor explícito o por defecto

El constructor explícito, denominado también *constructor de inicialización a ceros*, permite generar objetos limpios, libres de cualquier basura que el compilador pudiera darles al momento de realizar la asignación de memoria.

Este tipo de constructor es uno de los más utilizados y permite, al igual que el compilador por omisión en el caso de los objetos globales, inicializar con ceros las variables numéricas y a nulos las variables alfanuméricas, aunque también podríamos utilizarlo para que inicializara nuestros objetos con algún valor predefinido [1],[2],[3],[4],[5].

Su sintaxis general es la siguiente:

Clase( );

Ejemplo: *Persona( );*

Ejemplo práctico:

```
#include <iostream.h>
#include <string.h>
class Persona
{
  private:
        char nombre[30];
        char curp[20];
        int edad;
  public:
        Persona( );  //Declaración del constructor explícito
        void asignarDatos(char [30], char [20], int);
        void listarDatos( );
};
```

```
Persona( )::Persona( )
{
    strcpy(nombre, " ");
    strcpy(curp, " ");
    edad=0;
}
```

Cuerpo del constructor explícito o por defecto

```
void Persona::asignarDatos(char n[30], char c[20], int e)
{
    strcpy(nombre, n);
    strcpy(curp,c);
    edad = e;
}

void Persona::listarDatos( )
{
    cout<<"NOMBRE: "<<nombre<<endl;
    cout<<"CURP: "<<curp<<endl;
    cout<<"EDAD: "<<edad<<endl;
}
```

El programa sería:

```
#include <iostream.h>
#include <conio.h>
#include "Persona.h"

void main( )
{
    Persona   person1;    //Se ejecuta el constructor explícito

    person1.listarDatos( ); //Nos imprime un objeto de tipo persona, limpio y
                                                libre de basura
    person1.asignarDatos("Leonel Navarro", "NACL880613HHGVSN23",26);
    person1.listarDatos( ); //Imprime los datos recién asignados al objeto
                                                person1

    getch( );
}
```

Referencias:

[1] Ceballos, F.J. (1997), *Programación Orientada a Objetos con C++*, 2 ed., Alfaomega Ra-Ma.

[2] Deitel, P. & Deitel, H. (2008), *C++ Cómo Programar*, 6ta. ed., Pearson Prentice Hall.

[3] Joyanes, L. (2006), *Programación en C++. Algoritmos, estructuras de datos y objetos*, 2 ed., McGraw Hill.

[4] Joyanes, L. & Zahonero, I. (2010), *Programación en C, C++, Java y UML*, McGraw Hill.

[5] Smith, J. (2001), *Desarrollo de proyectos con programación orientada a objetos con C++*, Thomson Learning.