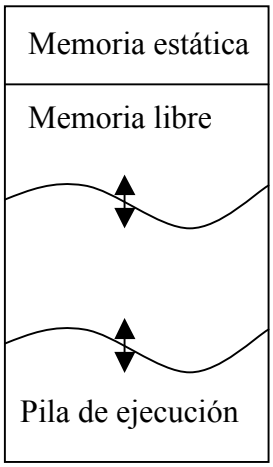
Unidad 1
MANEJO DE MEMORIA.

- **Memoria Estática.**
- **Memoria Dinámica.**

Modelo lógico de la memoria principal (figura 2.1), en el que se considera a la memoria como un arreglo unidimensional de bytes que está dividido en tres partes (memoria estática, memoria libre –heap o montón- y pila de ejecución) :



Memoria Alta

Memoria baja

Figura 2.1

Manejo de Memoria Estática: Término que aplicaremos principalmente al uso y manejo de variables, arreglos, etc., declarados de manera estática.

Una definición sencilla de arreglo sería: Colección ordenada de objetos, llamados elementos del arreglo, todos del mismo tipo.

En C++ un arreglo unidimensional se declara como:

```
int vec[5];          ← Declaramos un arreglo de 5 elementos de tipo entero.
```

El primer elemento del arreglo es vec[0], y el último es vec[4]. En la memoria de la computadora lo podríamos visualizar como:

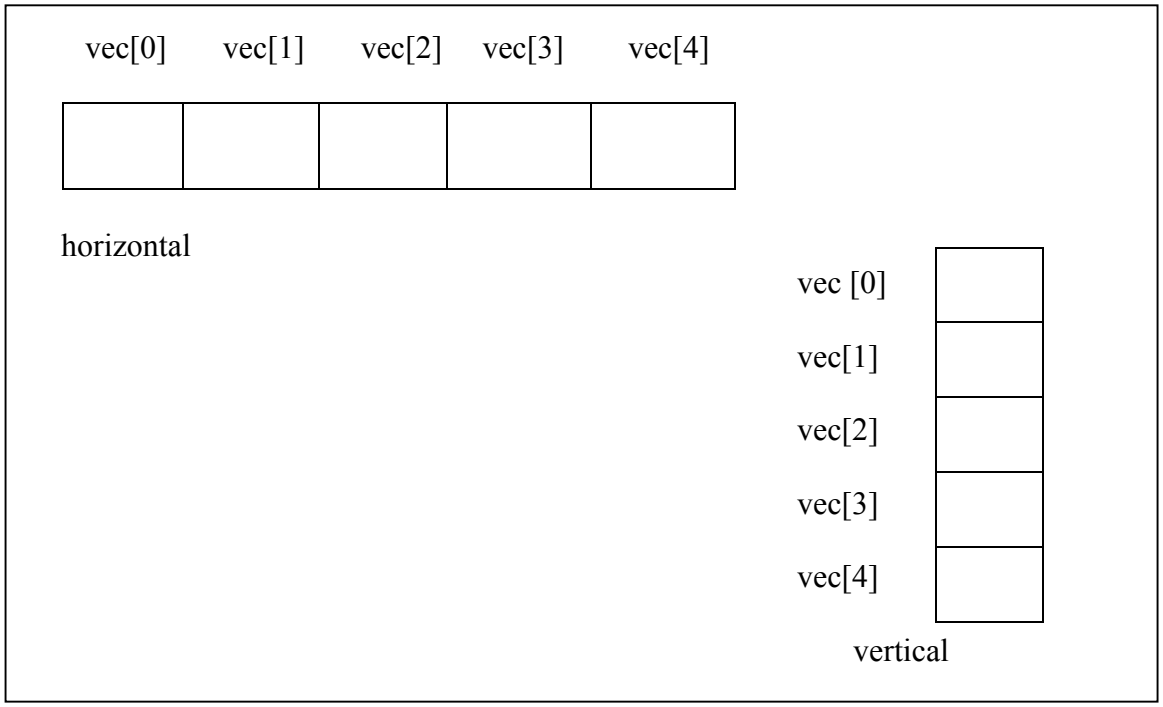


figura 2.2

float vec2[100]; ← Declaración de un arreglo de 100 elementos de tipo float.

char nombre[50]; ← Declaración de un arreglo de 50 elementos de tipo char. A este tipo de arreglos se les conoce comúnmente como cadena de caracteres, ya que se utilizan principalmente para almacenar cadenas.

Los arreglos multidimensionales se declaran de la siguiente forma:

- Arreglo Bidimensional:
  int mat[3][3]; ← Arreglo bidimensional de enteros de 3 X 3.

De forma gráfica lo podemos ver como:

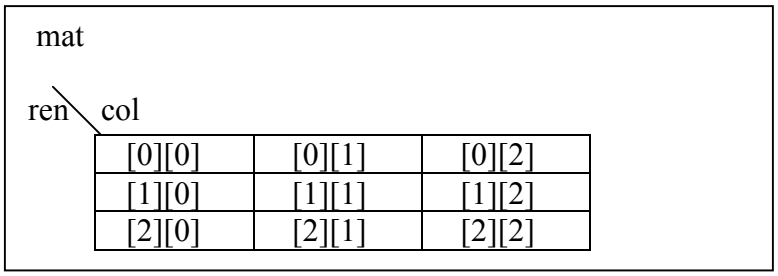


figura 2.3

Pero como la memoria física de las computadoras es lineal (ver figura 2.4), en realidad los elementos quedarían colocados fisicamente como:

mat (El nombre del arreglo representa la dirección de inicio).

[0][0]	[0][1]	[0][2]	[1][0]	[1][1]	[1][2]	[2][0]	[2][1]	[2][2]

Figura 2.4

float mat[2][7]; ← Arreglo bidimensional de tipo float de 2 X 7.

char nombre[10][30]; ← Arreglo de cadenas de caracteres.

C++ nos permite declarar arreglos de diferentes tamaños (multidimensionales), todo depende de la capacidad de memoria con que cuente nuestra PC, por ej:

La declaración de un arreglo tridimensional sería:

```
float  arreglo[3][3][3];
```

Los arreglos pueden ser inicializados al momento de declararlos, ejemplo:

```
int  vec[3]={1,2,3};
```

```
float  a[4] = {1.5, 2.5, 3.5, 4.5};


double d[3][3] = {{1.55, 2.55, 3.55},
                  {4.55, 5.55, 6.55},
                  {7.55, 8.55, 9.55}};

 char [3][10] = {"Alicia", "Maria", "Mario"};
```

Para comprender la teoría expuesta es mejor practicarla con ejemplos. Con los siguientes ejercicios veremos las facilidades que nos proporcionan los entornos de desarrollo (IDE) para escribir nuestros programas tanto en modo consola, así como en ambiente visual.

NOTA: *Las alternativas que se presentarán a lo largo del manual representan solo eso, ya que como todo en programación, existen diferentes formas de escribir nuestro código, siendo nuestra principal tarea practicar para así encontrar y establecer un estilo propio*

Práctica #1. Escribir una aplicación que utilice un arreglo unidimensional estático.

Como una primera alternativa vamos a calcular el promedio de 5 calificaciones.

- Definición de la clase VectorEst en el archivo VectorEst.h

```
#ifndef VectorEstH
#define VectorEstH
//---------------------------------------------------------------------------
enum {Max = 5}; // Declaramos un enumerado para definir el tamaño máximo de nuestro
                // arreglo, esto con el fin de aplicar la técnica del enum hack,  la cuál
                // evita que el compliador reserve espacio de memoria en la declaración de la
                // clase, ya que no estamos usando valores constantes, sino variables.
class VectorEst
{
  int vec[Max];
  public:
    void asignar(int pos, int cal);
    int leer(int pos);
    //Practicamos la sobrecarga de funciones
    //Pase de la referencia
    void calcular_promedio(float &prom);
```

```
    //Pase por puntero
    void calcular_promedio(double *prom);
    //metodo que retorna un valor
    float calcular_promedio();
};
#endif
```

- Implementación de la clase VectorEst en el archivo VectorEst.cpp

```
//---------------------------------------------------------------------------
#pragma hdrstop

#include "VectorEst.h"

//---------------------------------------------------------------------------

#pragma package(smart_init)

void VectorEst::asignar(int pos, int cal)
{
 //Validacion del rango del arreglo
   if(pos >= 0 && pos < Max)
     vec[pos]=cal;
}
int VectorEst::leer(int pos)
{
 //Validacion del rango del arreglo
 //if(pos >= 0 && pos < Max)
   return vec[pos];
}

void VectorEst::calcular_promedio(double *prom)
{
   *prom = 0;
   for(int i = 0; i < Max; i++)
      *prom += vec[i];
   *prom/= Max;
}
```

```
void VectorEst::calcular_promedio(float &prom)
{
   prom = 0;
   for(int i = 0; i < Max; i++)
      prom += vec[i];
   prom/= Max;
}

float VectorEst::calcular_promedio()
{
   float prom = 0;
   for(int i = 0; i < Max; i++)
      prom += vec[i];
   prom/= Max;
   return prom;
}

//---------------------------------------------------------------------------
```

- Escribimos una pequeña aplicación

```
//---------------------------------------------------------------------------
#include <vcl.h>
#include <iostream.h>

#include "VectorEst.h"
#pragma hdrstop
//---------------------------------------------------------------------------
/* Para agilizar la prueba de nuestra aplicación usaremos
   la alternativa de generar datos de manera aleatoria.

   También para probar nuestros métodos sobrecargados
   habilitaremos y deshabilitaremos la invocación al
   cálculo del promedio con las diferentes alternativas que
   escribimos.
*/
```

```
int main(int argc, char* argv[])
{
    VectorEst vec1;
    randomize();//Funcion que inicializa el generador de numeros aleatorios
    for(int i = 0 ; i < Max ; i++)
        vec1.asignar(i,random(101));
                //random genera nums. aleatorios entre 0 y n-1
    cout<<"\t Datos Asignados al Vector : "<<endl<<endl;
    for(int i = 0 ; i < Max ; i++)
        cout<<"  vec["<<i<<"] : "<<vec1.leer(i)<<endl;
    //Habilitamos el código del método que queremos usar

    /*
    float promedio;
    vec1.calcular_promedio(promedio);
    cout<<"\n Promedio : "<<promedio;
    */

    double promedio;
    vec1.calcular_promedio(&promedio);
    cout<<"\n Promedio : "<<promedio;

    //Otra alternativa seria:
    //cout<<"\nPromedio : "<<vec1.calcular_promedio();

    cin.get();

    return 0;
}


//---------------------------------------------------------------------------
```

Práctica #2. Reescriba el programa anterior pero utilizando programación visual.

Ahora vamos a migrar nuestro programa al entorno visual, en el cual aprovecharemos las ventajas que nos ofrece C++ Builder para escribir aplicaciones Windows. En estas aplicaciones debemos tener siempre presentes los requisitos que debemos respetar como:

- Un correcto orden en la colocación de nuestros componentes.
- Realizar las suficientes validaciones en los componentes que utilizemos.
- etc.

Por lo que es necesario realizar un buen diseño antes de empezar a agregar componentes sin siquiera tener una idea de lo que buscamos o queremos realizar.

figura 2.5

Componentes a utilizar:

Componente	Propiedad	Asignar
Forma	Caption	Promedio de 5 calificaciones
Panel1	Align Color	alBottom clGrid
Button1	Caption Enabled	Generar Calificaciones Trae
Button2	Caption Enabled	Calcular Promedio False
Button3	Caption	Otro ?
Button4	Caption	Acerca De
Button5	Caption	Salir

En esta aplicación nos vamos a auxiliar de un arreglo unidimensional para almacenar los datos generados de forma aleatoria, y así en otro componente poder utilizarlos.

El arreglo auxiliar lo colocaremos en la sección private de la clase forma, por lo que debemos abrir su archivo cabecera (en nuestro caso AppVectorEst.H). Como el tamaño del arreglo esta definido por la variable de tipo enumerado Max, la cual fue declarada en el archivo cabecera de la implementación de la clase (VectorEst.h), debemos incluirlo en este archivo, y con esto nos ahorramos el incluir el archivo cabecera de nuestra clase en el archivo de la aplicación.

```
#ifndef AppVectorEstH
#define AppVectorEstH
//---------------------------------------------------------------------------
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
#include <ExtCtrls.hpp>
#include "VectorEst.h"          <-------------------------- Aquí
//---------------------------------------------------------------------------
class TForm1 : public TForm
{
__published:   // IDE-managed Components
        TPanel *Panel1;
        TButton *Button1;
        TButton *Button2;
```

```
        TButton *Button3;
        TButton *Button4;
        TButton *Button5;
        void __fastcall Button1Click(TObject *Sender);
        void __fastcall Button2Click(TObject *Sender);
        void __fastcall Button3Click(TObject *Sender);
        void __fastcall Button4Click(TObject *Sender);
        void __fastcall Button5Click(TObject *Sender);
private:        // User declarations
        int vecaux[5];
public:         // User declarations
        __fastcall TForm1(TComponent* Owner);
};
extern PACKAGE TForm1 *Form1;
#endif
```

Aquí

Escribe el siguiente código en su correspondiente componente:

```
#include <vcl.h>
#pragma hdrstop

#include "AppVectorEst.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
//---------------------------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  randomize();
  int x=65, y = 10;
  Canvas->TextOut(10,y,"Calificaciones Generadas : ");
  y= 30;
```

```
  for(int i = 0; i < Max; i++)
  {
    vecaux[i] = random(101);  //generamos los datos aleatorios y los almacenamos
                              //en el arreglo vecaux
    Canvas->TextOut(x,y,vecaux[i]);
    y+= Canvas->TextHeight('T');
  }
}
//---------------------------------------------------------------------------


void __fastcall TForm1::Button2Click(TObject *Sender)
{
  VectorEst vec1;
  for(int i = 0; i < Max; i++)
     vec1.asignar(i,vecaux[i]);
  AnsiString cad;
  cad.sprintf("Promedio : %3.2f",vec1.calcular_promedio());
  Canvas->TextOutA(10,100,cad);
}


void __fastcall TForm1::Button3Click(TObject *Sender)
{
  Repaint(); // método para borrar o limpiar la forma
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Button4Click(TObject *Sender)
{
   ShowMessage("Nombre :        \nMateria :     \nFecha : ");
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Button5Click(TObject *Sender)
{
    Close();
}
```

La clase VectorEst que diseñamos y probamos en modo consola, es la misma que utilizaremos en ambiente visual. Solo copia los archivos .H y . CPP de la clase diseñada a la carpeta donde estas trabajando con el proyecto visual. Después selecciona la opción Project, luego Add Project y selecciona el archivo CPP de tu clase y listo. Después compila y si no hubo errores ejecuta tu aplicación.

Práctica #3: Construye otra aplicación en modo visual (ver figura 2.6), de tal forma que captures las calificaciones en un componente Edit y, estas se vayan insertando en un componente StringGrid, uno por uno, y después calcula su promedio.

figura 2.6

Manejo de Memoria Dinámica: En C++ la creación dinámica de variables se lleva a cabo utilizando el operador **new** y estas variables dinámicas se eliminan empleando el opeador **delete** . Con estos operadores el programador tiene la libertad de reservar y eliminar espacio de memoria cuando lo necesite.

Una variable creada con el operador ***new*** dentro de cualquier bloque, perdura hasta que es explícitamente borrada con el operador ***delete***. Puede traspasar la frontera de su bloque y ser manipulada por instrucciones de otros bloques.

Un aspecto diferente con la función ***malloc()***, que es el método más utilizado para reservar dinámicamente memoria en ANSI C, es que ésta devuelve un ***puntero a void*** (*void) que es después convertido al tipo de variable que se desea. Esa conversión se evita con ***new***, eliminando así una posible fuente de problemas.

Se puede utilizar el operador ***new*** para crear variables de cualquier tipo, este devuelve, en todos los casos, un puntero a la variable creada. También se pueden crear variables de tipos definidos por el usuario:

```
enum {N_DIGITOS=9};
enum {N_CAL = 5};
class Alumno
{
   char *nombre;
   char no_cont[N_DIGITOS];
   int califs[N_CAL];
    .
    .
    .

};

Alumno *Lista;
Lista = new Alumno[cuantos];
assert(Lista);   //Realizaremos un aserción o cualquier otra técnica que nos permita verificar
                 //si se pudo asignar la memoria dinámicamente, en caso contrario debemos
                 //abortar el programa o encontrar una solución al problema.
```

Cuando una variable ya no es necesaria se destruye con el operador ***delete*** para poder utilizar la memoria que estaba ocupando, mediante una instrucción del tipo:

```
delete []Lista;
```

Los siguientes ejemplos presentan la forma de reservar memoria dinámica

1).- Arreglo de caracteres:

```
#include <iostream.h>
#include <string.h>
#include <assert.h>

void main()
{
  char Nombre[50];
  cout << "Introduzca su Nombre:";
  cin.getline(Nombre, sizeof(Nombre));
```

```
  char *CopiaNombre = new char[strlen(Nombre)+1];
  assert(CopiaNombre);
  // Se copia el Nombre en la variable CopiaNombre
  strcpy(CopiaNombre, Nombre);
  cout << CopiaNombre;
  cin.get();
  delete [] CopiaNombre;
}
```

2).- Creación de una arreglo bidimensional (Matriz) de tipo double

```
#include <iostream.h>
#include <assert.h>
#include <stdlib.h>

int main()
{
 int n_ren,n_col, i, j;
 cout<<"No. De Renglones : ";
 cin>>n_ren;
 cout<<"No. De Columnas  : ";
 cin>>n_col;
 double **mat;
 // se reserva memoria para el vector de punteros
 mat = new double*[n_ren];
 assert(mat != 0);

 // se reserva memoria para cada renglón
 for (i=0; i < n_ren; i++)
 {
    mat[i] = new double[n_col];
    assert(mat[i]);
 }
 randomize();
 // se inicializa toda la matriz
 for(i=0; i < n_ren; i++)
   for(j=0; j < n_col; j++)
      mat[i][j]=random(1000)/.77;

 // se imprime la matriz
```

```
  for(i=0; i < n_ren ; i++){
  {
     for(j=0; j < n_col; j++)
       cout << mat[i][j] << "\t";
     cout << endl;
  }
....// liberamos la  memoria
  for(i=0; i < n_ren; i++)
   // se borran los renglones de la matriz
          delete [] mat[i];
  // se borra el vector de punteros
  delete [] mat;
  cin.get();
  return 0;
}
```

Práctica # 4. En este ejemplo además de utilizar los operadores new y delete, vamos a practicar la sobrecarga de operadores, el uso de las plantillas (template) y el concepto de inicializadores (alternativa que proporciona C++ para inicializar datos miembro de una clase).

- Versión modo consola.

```
//Definición de la clase Vector

#ifndef VectorH
#define VectorH


//---------------------------------------------------------------------------
template <class T>
class Vector
{
   T *vec;
   int tam;

  public:
```

```
      //Constructor general con argumento por defecto
    Vector(const int n=10);
    //Constructor de copia
    Vector(const Vector<T> & v_aux);
    // Destructor
    ~Vector();
    //Sobrecarga de operadores:
    //Acceso a elementos
    T & operator[](const int indice) const;
    //asignacion
    Vector<T> & operator=(const Vector<T> &v_aux);
    unsigned int tamanio()const;
};
#endif

//Implementacion de la clase Vector

#include <assert.h>
#pragma hdrstop
#include "Vector.h"
#pragma package(smart_init)

//Constructor con argumentos por defecto,usamos inicializadores
template <class T>
Vector<T>::Vector(const int n )//:tam(n)
{
  tam = n;
   vec = new T[tam];
   //con la función assert verificamos si se reservo el espacio de memoria
  //solicitado, sino la función aborta el programa
   assert(vec);  //assert(vec != 0);
  //inicializamos a 0's el arreglo
   for(int i = 0; i < tam; i++)
      vec[i] = 0;
}
```

```
//Constructor de copia
template <class T>
Vector<T>::Vector(const Vector<T> & v_aux):tam(v_aux.tam)
{
 // asignacion de espacio para los nuevos elementos
 vec = new T[tam];
 assert(vec);
 //Copiamos los valores de v_aux
 for(int i = 0 ; i < tam; i++)
    vec[i] = v_aux[i];
}


// Destructor
template <class T>
Vector<T>::~Vector()
{
  //liberamos la memoria
  delete []vec;
}


//Operador para accesar  elementos
template <class T>
T & Vector<T>::operator[](const int indice) const
{
  //retorna el elemento referido por indice
  if(indice >= 0 && indice < tam)
    return vec[indice];
  else //Opcional: si indice esta fuera del rango
    return vec[0];//regresamos el elemento 0
                //o abortamos el programa  ????
//Otra alternativa sería:
//    assert(indice >= 0 && indice < tam);
//    return vec[indice];
}
```

```
// operador de asignacion
template <class T>
Vector<T> & Vector<T>::operator=(const Vector<T> &v_aux)
{
  //liberamos el espacio de los datos anteriores
  delete []vec;
  tam = v_aux.tam;
  //creamos espacio para los datos nuevos
  vec = new T[tam];
  assert(vec);
  for(int i = 0; i < tam; i++)
     vec[i] = v_aux.vec[i];
  return *this;
}

template <class T>
unsigned int Vector<T>::tamanio()const
{
  return tam;
}
```

- **Versión visual.** A continuación se muestra una de las tantas alternativas que podemos escribir para probar y utilizar nuestra clase vector en su versión dinámica

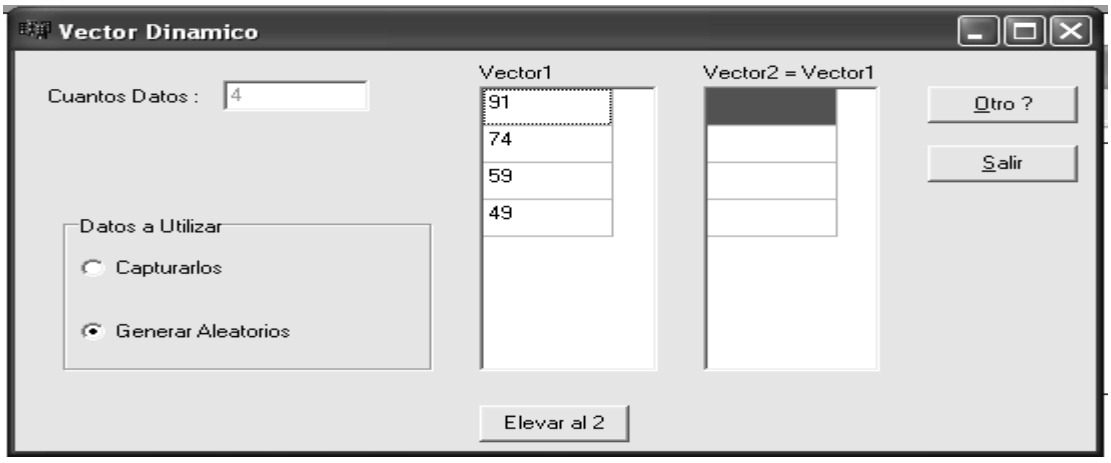


figura 2.7

Componentes a utilizar:

Componente	Propiedad	Asignar
Form1	Caption	Vector Dinámico
Label1	Caption	Cuantos Datos
Edit1	Text Hint ShowHint Seleccionar el evento KeyPress y escribir el código que se presenta.	Borrar el texto Presiona Enter true
Label2	Caption Visible	Introduce un valor y presiona Enter. False
RadioGroup1	Caption Ítems	Datos a utilizar -Capturarlos -Generar Aleatorios
StringGrid1 StringGrid2	FixedRows FixedCols RowCount ColCount Options (Solo StringGrid1) goEditing	0 0 10 1 true
Label3	Caption	Vector1
Label4	Caption	Vector2 = Vector1
Button1	Caption	Elevar al 2
Button2	Caption	&Otro ?
Button3	Caption	&Salir

// Código de la aplicación

```
#include <vcl.h>
#include "Vector.h"
#include "Vector.cpp"
#pragma hdrstop
```

```
#include "AppVec.h"
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1KeyPress(TObject *Sender, char &Key)
{
  if(Key == 13)
  {
    Edit1->Enabled = false;
    Label2->Visible= false;
    //Si escriben un valor no esperado, usamos 10 como default
    Edit1->Text=Edit1->Text.ToIntDef(10);
    StringGrid1->RowCount = Edit1->Text.ToInt();
    StringGrid2->RowCount = Edit1->Text.ToInt();
    RadioGroup1->SetFocus();
  }
  else
    Label2->Visible = true;
}
void __fastcall TForm1::RadioGroup1Click(TObject *Sender)
{
 StringGrid1->Enabled = true;
 if(RadioGroup1->ItemIndex == 0)
 {
  StringGrid1->SetFocus();
 }
 else
 {
  StringGrid1->SetFocus();
  randomize();
  for(int i = 0; i < Edit1->Text.ToInt(); i++)
      StringGrid1->Cells[0][i]= random(100);
 }
}
```

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
   int n = Edit1->Text.ToInt();
    Vector<int> v1(n);
    for(int ren = 0; ren < n; ren++)
       v1[ren] = StringGrid1->Cells[0][ren].ToInt();
    for(int ren = 0; ren < n; ren++)
    {  //multiplicamos cada dato por si mismo
       v1[ren] *= v1[ren];
       StringGrid1->Cells[0][ren]= v1[ren];// * v1[ren];
    }
    //Probamos el constructor de copia  y lo mostramos en StringGrid2
    Vector<int> v2(v1); //
    for(int ren = 0; ren < n; ren++)
       StringGrid2->Cells[0][ren]=v2[ren];
}

void __fastcall TForm1::Button2Click(TObject *Sender)
{
  for(int r = 0; r < Edit1->Text.ToInt();r++)
  {
    StringGrid1->Cells[0][r]="";
    StringGrid2->Cells[0][r]="";
  }

  Edit1->Enabled=true;
  Edit1->Clear();
  StringGrid1->RowCount = 10;
  StringGrid2->RowCount = 10;
  RadioGroup1->ItemIndex = -1;
}

void __fastcall TForm1::Button3Click(TObject *Sender)
{
  Close();
}
```

No olvides incluir en la carpeta donde guardes tu aplicación, los archivos .h y .cpp de la clase vector, agregarlos a tu proyecto (Ir a la opción Project/ addProject, seleccionar los archivos a incluir y listo). Recuerda que estamos probando la misma clase que escribimos y ya utilizamos en modo consola.