

# Interconectividad entre Lenguajes de Programación

**Autores:**

Salazar Ramírez Norman Francisco

Cota Ortiz Maria de Guadalupe

Flores Pérez Pedro

# INDICE

---

	<b>Página</b>
<b>Presentación. ....</b>	<b>4</b>
<b>Dinámic Link Libraries.....</b>	<b>5</b>
<b>Registro manual de una dll, un ActiveX o un ocx en Windows.....</b>	<b>5</b>
<b>Rutinas de Código Ensamblador intercaladas en Lenguaje C.....</b>	<b>6</b>
<b>Creación de una dll en Borland C++.....</b>	<b>8</b>
<b>Creación de una dll en Visual C++.....</b>	<b>12</b>
<b>Creación de Dll's con Visual Basic para Visual Basic.....</b>	<b>17</b>
<b>Secciones del archivo de definición.....</b>	<b>19</b>
<b>Llamadas a funciones de las dll desde Visual Basic.....</b>	<b>21</b>
<b>Pasar valores por referencia hacia las funciones de la dll que se ejecutarán...</b>	<b>23</b>
<b>Paso de parámetros usando cadenas de caracteres.....</b>	<b>24</b>
<b>Conclusiones.....</b>	<b>27</b>

# INTRODUCCIÓN

En este trabajo se desarrolla el tema que permite identificar el procedimiento de ligado o incrustación de código escrito en distintos lenguajes de programación y distintos niveles de máquina (ver Figura No. 1), y se describen los procedimientos para incrustar código C con Ensamblador, crear una Dll con Borland 5.02, incluir código C en librerías generadas por el compilador de Visual C++, y la forma de utilizar las dll's creadas con Visual Basic y Java.

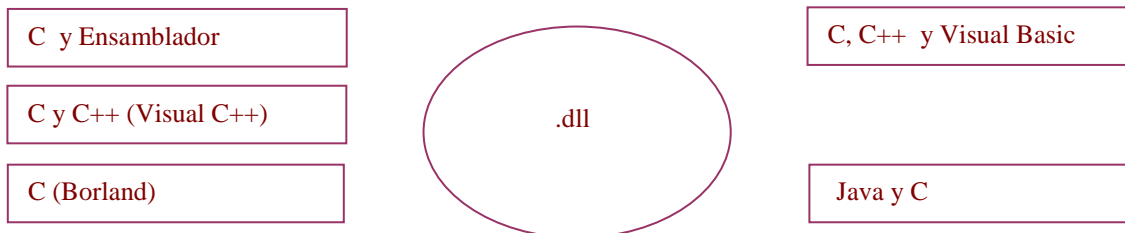


Figura No. 1

Además se analizan las diferencias que existen en el paso de argumentos o valores de retorno de una función cuando se utilizan valores enteros y cadenas de caracteres, ya que la interpretación semántica de los mismos es distinta en cada lenguaje, pues esto depende de la máquina virtual diseñada para cada uno de ellos.

# PRESENTACIÓN

Para utilizar eficientemente la tecnología computacional, es necesario optimizar el uso de los recursos de hardware y software, y contar con personal altamente capacitado y actualizado en dicha materia.

Entre las principales alternativas en materia de programación orientadas a la administración de la memoria que consumen las aplicaciones, se ha distinguido la que permite particionar un programa en módulos, que se cargan en memoria en forma dinámica cuando son invocados por alguno de los módulos restantes, y se descargan una vez que han cumplido con su función. A este procedimiento se le conoce como método de ligado (“link”). Con este tipo de programación se logra eficientar el uso de la memoria y aumentar la rapidez de la aplicación en tiempo de ejecución.

Por otra parte, estos componentes tienen la característica de poder programarse con distintos lenguajes de programación, mismos que son seleccionados de acuerdo al nivel de complejidad de la aplicación, y su ejecución es totalmente ‘transparente’ para el usuario que utiliza la utiliza (ver figura No. 2).

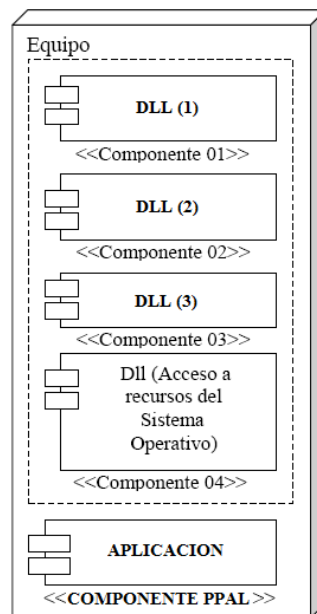


Fig. 2.- Estructura de componentes de un programa que incluye Librerías de Enlace Dinámico (DLL)

## DINAMIC LINK LIBRARY

Una Librería de Enlace Dinámico “**DLL**” (Dynamic Link Library), es un archivo binario que contiene funciones llamados ‘puntos de acceso’, los cuales pueden ser invocados en tiempo de ejecución por múltiples aplicaciones, descargándose una vez que han cumplido con su objetivo.

### REGISTRO MANUAL DE OBJETOS COMO DLL’s, ACTIVEX O UN OCX EN WINDOWS

Algunos de los errores que se registran en equipos con ambiente Windows cuando se ejecuta alguna aplicación, se deben a que alguna Dll u otro tipo de objetos de este tipo no están debidamente registrados en el sistema. Regularmente esto suele suceder cuando se trata de ejecutar algún objeto que requiere acceso a datos (RDO350.DLL) o con los controles de tipo ActiveX (ficheros con la extensión .ocx).

Para resolver problemas de este tipo, se pueden registrar estos archivos manualmente utilizando la herramienta **Regsvr32**, aplicando el siguiente procedimiento:

`Regsvr32 [/u] [/s] <nombre del fichero>`

Donde los parámetros opcionales [/u] [/s] significan lo siguiente:

[/u] – se utiliza para eliminar registros de una DLL o un .OCX.

[/s] - modo "silencioso" - no despliega los mensajes durante la operación.

#### Por ejemplo:

En Menú Inicio, elegir opción de “Ejecutar”

Escribir la instrucción: `REGSVR32 c:\windows\system\Dao350.dll`

Una vez hecho esto, aparecerá algún mensaje indicando si el resultado obtenido fue fallido o exitoso.

## RUTINAS DE CÓDIGO ENSAMBLADOR INTERCALADAS CON LENGUAJE C

Hay ocasiones en que es necesario acceder a alguno de los recursos del equipo en forma directa, y ésto sólo puede lograrse utilizando el lenguaje de más bajo nivel que está al alcance de un programador, que es el Lenguaje Ensamblador.

Esta posibilidad permite obtener rutinas óptimas que pueden integrarse fácilmente en un proyecto donde se utilizan lenguajes de programación de medio y alto nivel.

A continuación se presentan algunos ejemplos de código C con ensamblador para realizar operaciones simples que permitan observar su funcionamiento. Estas rutinas pueden compilarse con cualquier compilador de C/C++.

### Ejemplo para obtener el producto de dos números introducidos por el usuario

```
#include <stdio.h>
#include <conio.h>
void main(void ) {
    int dato1, dato2, resultado;
    clrscr();
    printf(" Proporciona dos números para multiplicarlos: ");
    scanf("%d %d", &dato1, &dato2);
    asm { push ax; push cx; }
    asm mov cx,dato1
    asm mov ax,0h
    Etiqueta:
        asm add ax,dato2
        asm loop Etiqueta
        asm mov resultado,ax
    asm { pop cx; pop ax; }

    printf("Su producto es: %d", resultado);
    getch();
}
```

En el ejemplo anterior con la instrucción 'push' se guarda en la pila los valores de los registros utilizados, y se restauran al final con la instrucción 'pop' para no restaurar los valores que se encuentran en la pila. De igual forma al principio se inicializa CX con el valor del primer dato introducido, y AX con 0, ya que es el registro que permitirá acumular resultados

En dicho código un ciclo es representado a partir de la etiqueta hasta la instrucción LOOP, donde se suma el segundo número introducido tantas veces como se indique en CX. La instrucción LOOP decrementa en 1 CX en cada vuelta del ciclo y finaliza cuando CX = 0. Con ello obtenemos el resultado que es equivalente al producto de CX y AX.

En el próximo ejemplo se muestran los contenidos de los registros AX, BX, CX y DX. Las líneas que inician con '//' se aplican como comentarios.

## Ejemplo que muestra el contenido de los Registros de Propósito General Internos del Procesador

```
#include <stdio.h>
#include <conio.h>
void main(void) {
    int regax, regbx, regcx, regdx;
    regax = 0; regbx = 0; regcx = 0; regdx = 0;
    clrscr();
    asm {
        push ax;
        push bx;
        push cx;
        push dx;
    }
    // asm mov cx, 02h
    // asm mov dx, 04h
    asm mov regax, ax
    asm mov regbx, bx
    asm mov regcx, cx
    asm mov regdx, dx

    asm {
        pop ax;
        pop bx;
        pop cx;
        pop dx;
    }
    printf("Valor de AX: %d", regax);
    printf("Valor de BX: %d", regbx);
    printf("Valor de CX: %d", regcx);
    printf("Valor de DX: %d", regdx);
    getch();
}
```

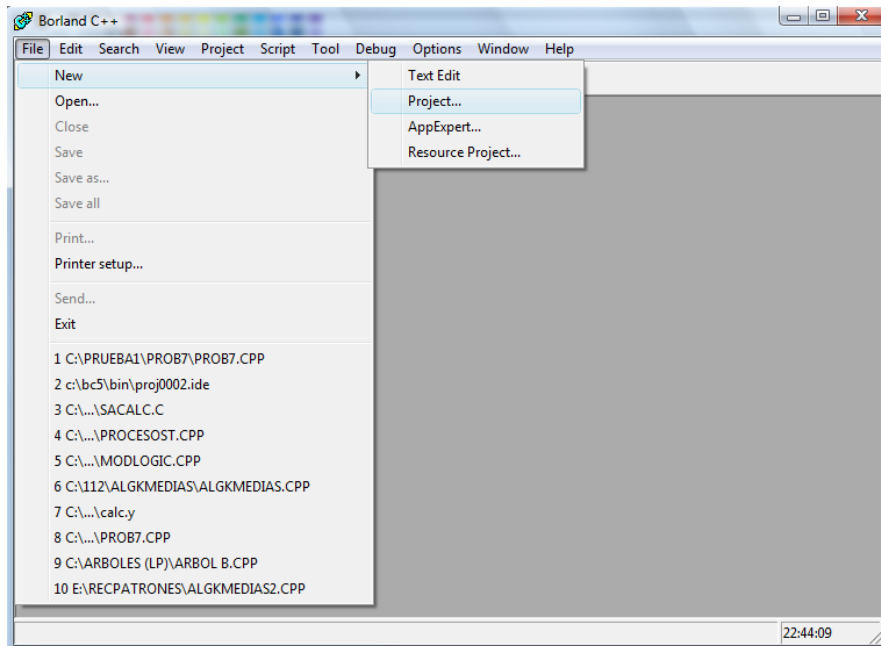
**Nota:** Eliminando las líneas de comentarios se modifican los valores de CX y DX antes de mover sus valores a las variables correspondientes.

# CREACIÓN DE UNA DLL EN BORLAND C++

Los pasos a seguir para crear una DLL (Dinamic Link Library), en el compilador Borland C++ versión 5.02, usando como ejemplo la función ‘suma’ que regresa un valor de tipo entero, son los siguientes:

## 1) Crear un proyecto:

Lo primero que se debe hacer es crear un proyecto ‘nuevo’, lo cual se logra seleccionando la opción File- >New -> Project. Cabe aclarar que el nombre que se fije para el proyecto será el nombre que se asigne a la DLL que se va a generar.

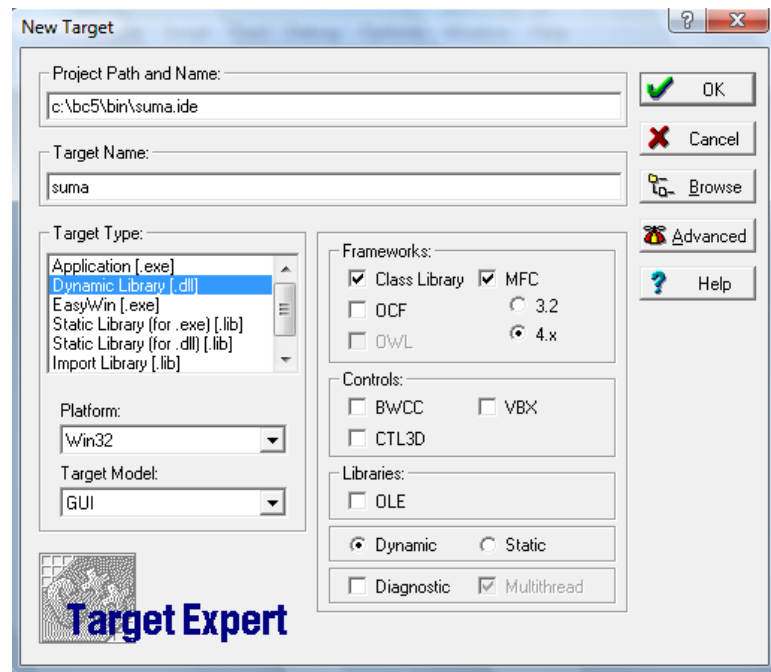


En el dialogo ‘*New Target*’ que se presentará se deben realizar las siguientes acciones:

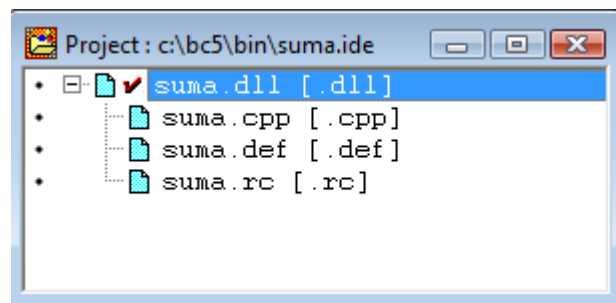
- a) En el apartado de ‘*Project Path and Name*’ se escribe la ruta y el nombre que tendrá el proyecto (la extensión del proyecto es ‘.ide’). El nombre que se escriba aquí automáticamente se escribirá en el apartado de ‘*Target Name*’.
- b) En ‘*Target Type*’ se deben fijar los siguientes parámetros:
  - Seleccionar Dinamic Library [.dll] en la lista.
  - Elegir Win32 en *Plataform*.
  - Elegir GUI en *Target model*.



- c) En '*Frameworks*', es necesario seleccionar las casillas de 'Class Library' y de 'MFC'. En caso de que no se puedan seleccionar es necesario quitar la selección a OWL.
- d) La opción de 'Dynamic' debe ser elegida en el apartado de 'Libraries'.
- a) Por ultimo se aceptan las opciones elegidas presionando *OK*.



## 2) Identificar archivos de proyecto:



## 3) Crear el código del proyecto:

Abrir el archivo con terminación '*.cpp*' para escribir el código fuente.

El código no debe contar con una función 'main()', y debe estar conformado solamente por funciones llamadas 'puntos de acceso' a la DLL. Además no es necesario que todas las funciones sean declaradas como 'externas', sino que pueden existir módulos que pueden ser invocados a través de los puntos de acceso.

El procedimiento para declarar funciones externas es el siguiente:

- a) Se escribe la palabra reservada **extern "C"** y se abren llaves '{'. Todas las funciones que estén dentro de estas llaves serán reconocidas como externas o puntos de acceso a la DLL.
- b) Se escriben las funciones que se desee que sean externas, pero a diferencia de una función normal, la palabra reservada **\_stdcall** debe ser colocada entre el tipo de dato que regresa y el nombre de la función.  
El propósito de **\_stdcall** es generar funciones que usen un método llamado estándar.
- c) Se cierran las llaves '}'.

La .DLL de ejemplo solo tiene una función y después de seguir los pasos anteriores se vería de la siguiente forma:

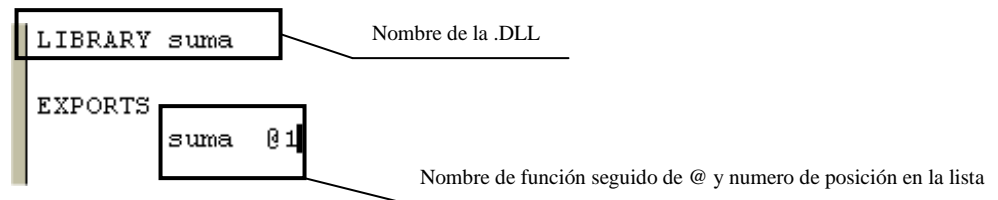
```
extern "C" {  
    int _stdcall suma() {  
        int resultado, valor1, valor2;  
        valor1=10;  
        valor2=10;  
        resultado=valor1 + valor2;  
        return resultado;  
    }  
}
```

Las funciones que se escriban después de esta llave serán externas .  
Se debe escribir en cada función dentro de extern.  
Se debe cerrar la llave de extern.

#### 4) Exportar funciones en el archivo de definición:


Este archivo, cuyo propósito es definir las funciones que se exportarán, lleva el mismo nombre del proyecto y la terminación .def, y debe incluir las siguientes especificaciones:

- a) Se escribe la palabra en mayúsculas LIBRARY seguido del nombre de la .DLL, dicho nombre debe ser el mismo que el del proyecto.
- b) Se escribe la palabra en mayúsculas EXPORT.
- c) Se escribe una lista con los nombres de las funciones a exportar. Después de cada nombre se escribe el símbolo '@' seguido de un numero autoincremental que inicie en '1', ya que es la forma de identificar la posición que tiene la función en la lista exportada.



### 5) Construir el Proyecto:

El ultimo paso para la creación de la .DLL es construir el proyecto. Esto se logra por medio de las siguientes formas:

- Seleccionando el icono .
  - Presionando las teclas Alt + F9
  - O seleccionado en el menú la opción Proyecto->Build All
- 6) Por último, una vez que es compilada la dll correspondiente, queda lista para ser utilizada por las aplicaciones para las cuales fue creada.

## CREACIÓN DE UNA DLL CON VISUAL C++

1. Crear una función suma en C que regrese valor de tipo entero.
2. Crear una Dll con Visual C++ con las siguientes instrucciones:

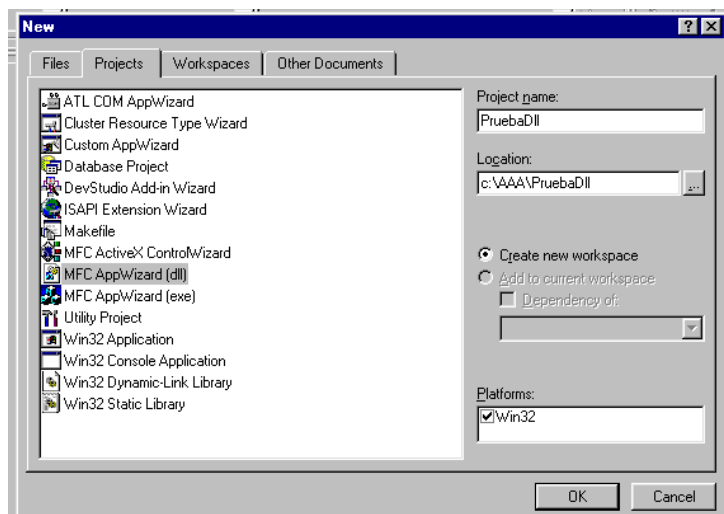
Crear un proyecto nuevo con la opción: *MFC AppWizard (dll)*

En el mismo diálogo escribir el nombre del proyecto en '*Project name*'

En el mismo diálogo escribir la ruta en donde se guardará el proyecto '*location*'

Elegir la plataforma en '*Platforms*' Win32

Aceptar opciones elegidas con '*Ok*'



Una vez realizado lo anterior, en la siguiente forma que aparece '*Step 1 de 1*':

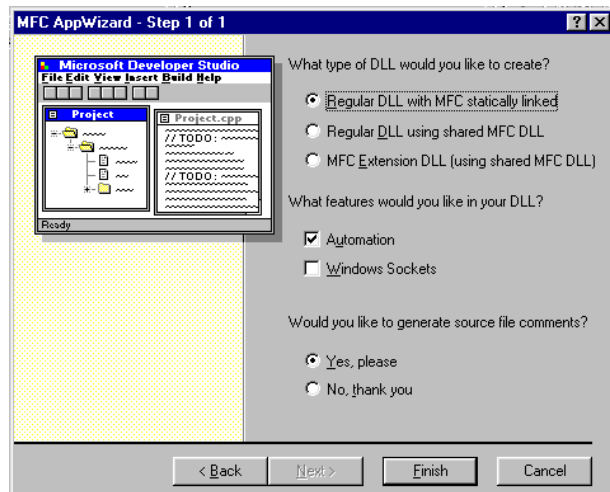
Elegir la opción de *Regular dll with MFC static linked*

Este tipo de DLL permite que las funciones creadas puedan ser llamadas por cualquier aplicación de tipo WIN32

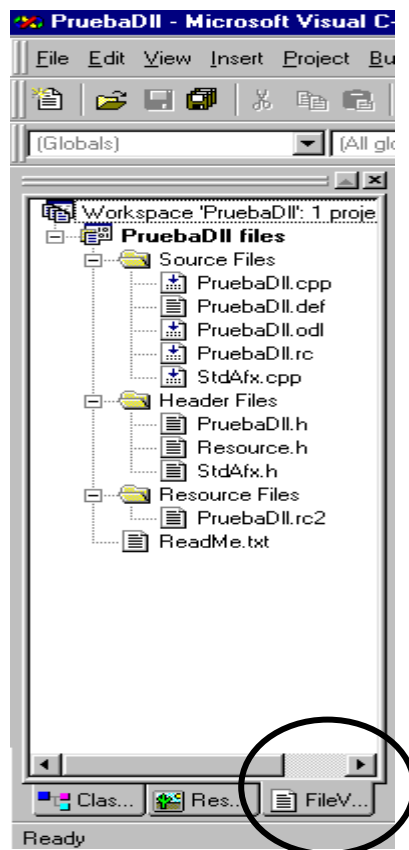
Seleccionar la opción de '*Automation*'

(Esta opción es una interface que permite manipular objetos implementados externamente con otras aplicaciones)

Seleccionar *'Yes, please'* para generar comentarios en el código como recursos adicionales



3. Localizar archivos de proyecto:



Localizar en *Source Files*:

Archivo con el mismo nombre del proyecto y extensión .cpp

Y agregar en la cabecera del archivo `#include "Prueba2.h"`, y la

Declaración como externa de la librería que se desea incluir de C con la ruta donde se encuentra, como a continuación se indica:

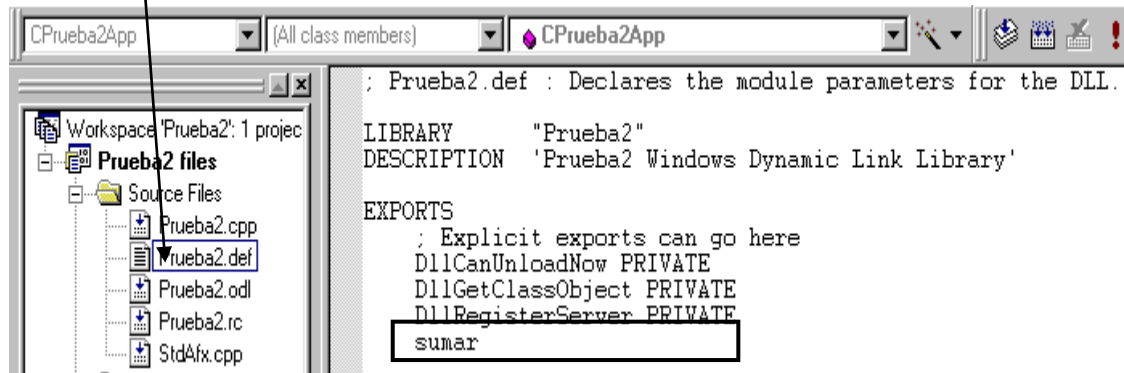
```
#include "stdafx.h"
#include "Prueba2.h"
//declaracion de funciones de c. codigo adicional mio
extern "C" {
#include "D:\dll\funcionc.h"
} // hasta aqui codigo adicional mio

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif
```

En el mismo archivo .cpp agregar función que invocará a la función de C

```
// CPrueba2App construction
CPrueba2App::CPrueba2App()
{
    // TODO: add construction code here
    // Place all significant initial
}
//código adicional mio
long CPrueba2App::sumar(void) {
    long resultado;
    // resultado = 20;
    resultado = suma();
    return resultado;
} // hasta aqui codigo adicional mio
```

En [Archivo Prueba2.def](#), agregar la declaración de la librería que se exportará para ser utilizado por otro lenguaje:

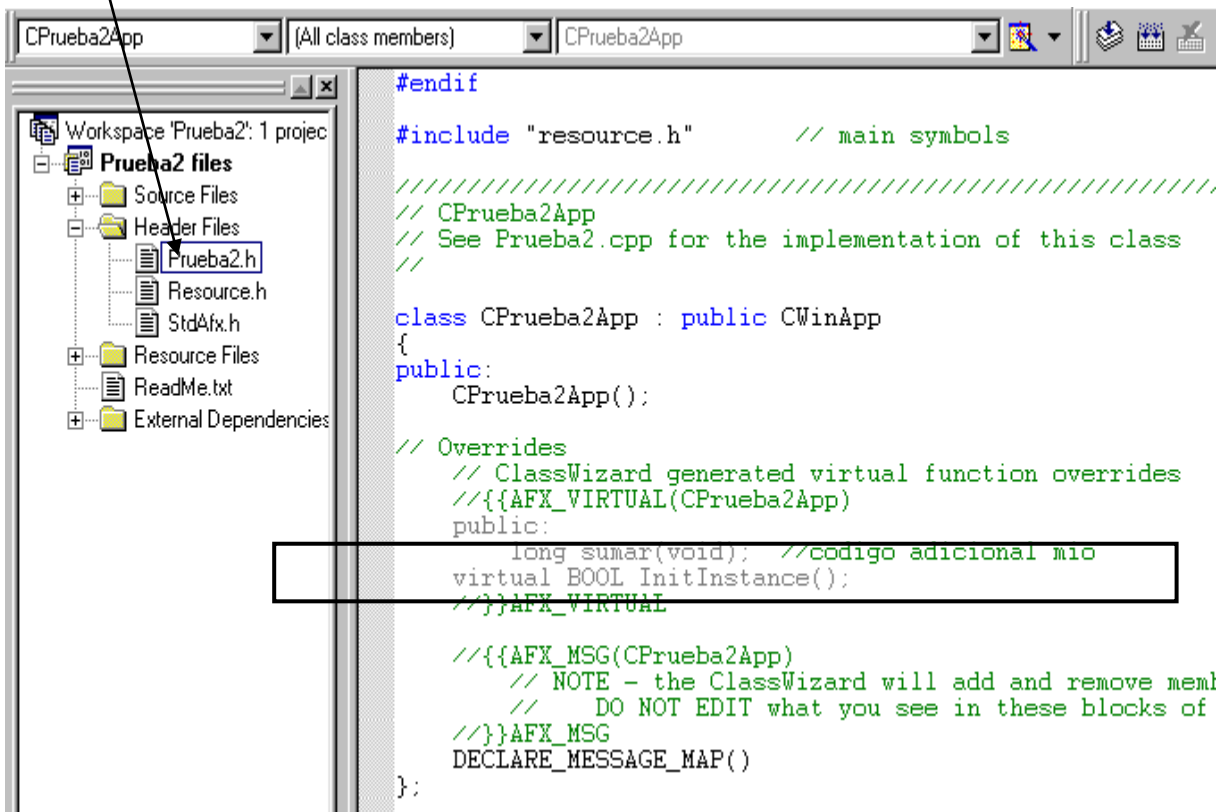


```
; Prueba2.def : Declares the module parameters for the DLL.

LIBRARY      "Prueba2"
DESCRIPTION  'Prueba2 Windows Dynamic Link Library'

EXPORTS
; Explicit exports can go here
DllCanUnloadNow PRIVATE
DllGetClassObject PRIVATE
DllRegisterServer PRIVATE
sumar
```

En [prueba2.h](#), agregar la función en la implementación de la clase:



```
#endif

#include "resource.h" // main symbols

////////////////////////////////////
// CPrueba2App
// See Prueba2.cpp for the implementation of this class
//

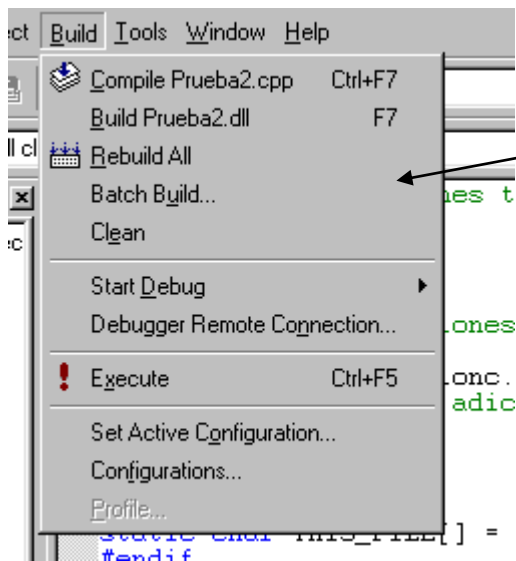
class CPrueba2App : public CWinApp
{
public:
    CPrueba2App();

// Overrides
// ClassWizard generated virtual function overrides
//{{AFX_VIRTUAL(CPrueba2App)
public:
    long sumar(void); //codigo adicional mio
    virtual BOOL InitInstance();
//}}AFX_VIRTUAL

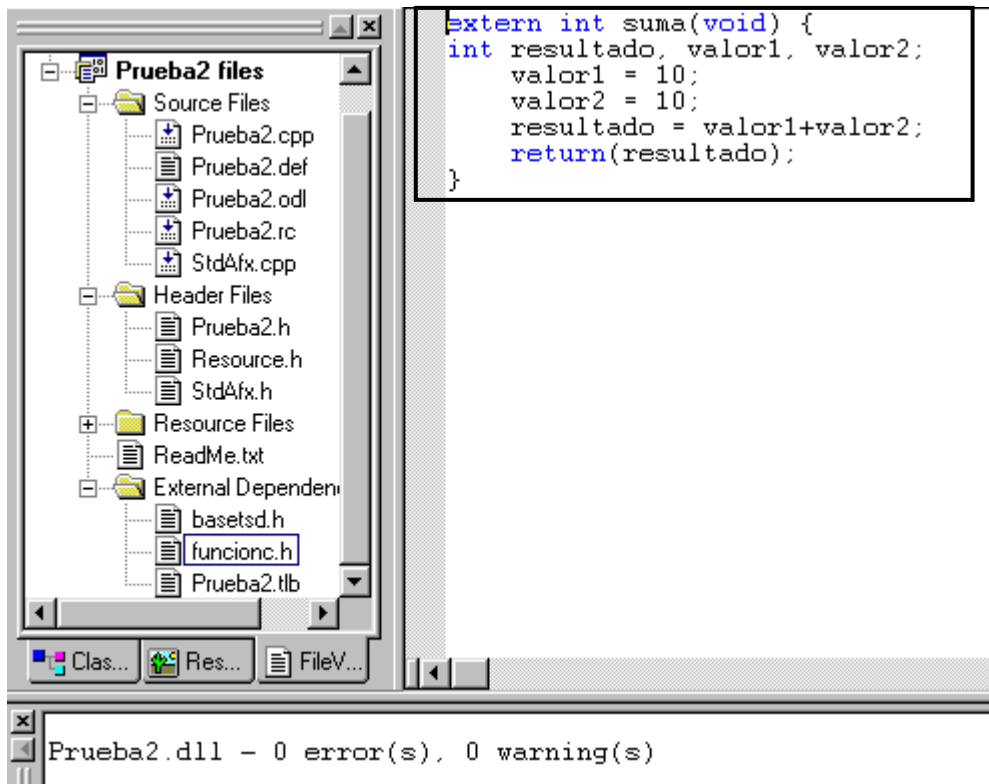
//{{AFX_MSG(CPrueba2App)
// NOTE - the ClassWizard will add and remove meml
// DO NOT EDIT what you see in these blocks of
//}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};
```

En el archivo [Readme.txt](#) se encuentran los comentarios generados explicando la función de cada archivo creado.

Enseguida se procede a compilar el proyecto para obtener la dll correspondiente



Al compilarse el proyecto, automáticamente se crea una nueva carpeta, en el proyecto llamado



“External Dependences”, donde se incluye el archivo `funcionc.h` que contiene las librerías de C.



# CREACIÓN DE DLL'S CON VISUAL BASIC PARA VISUAL BASIC

1. Crear un nuevo proyecto tipo Dll ActiveX



Por defecto se crea un recurso llamado Class1, que es un modulo de clase que contendrá las funciones de la DLL.

Para uso de formularios se deben fijar las opciones de:

- BorderStyle = 3 Fixed Dialog
- StartUpPosition = 2 CenterScreen

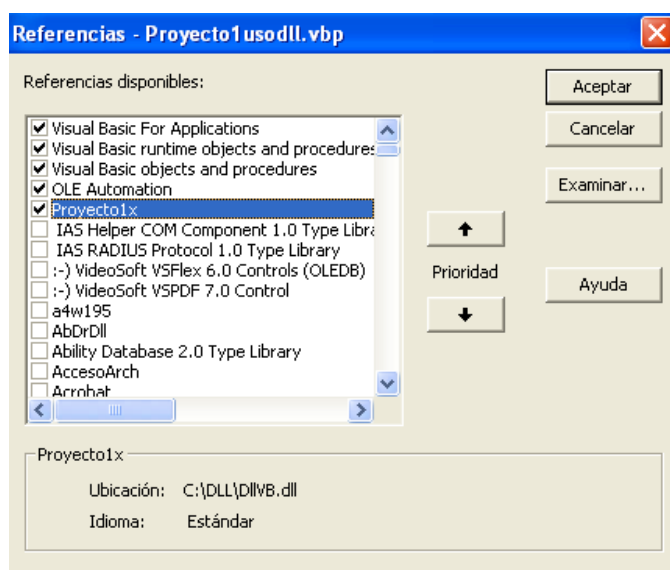
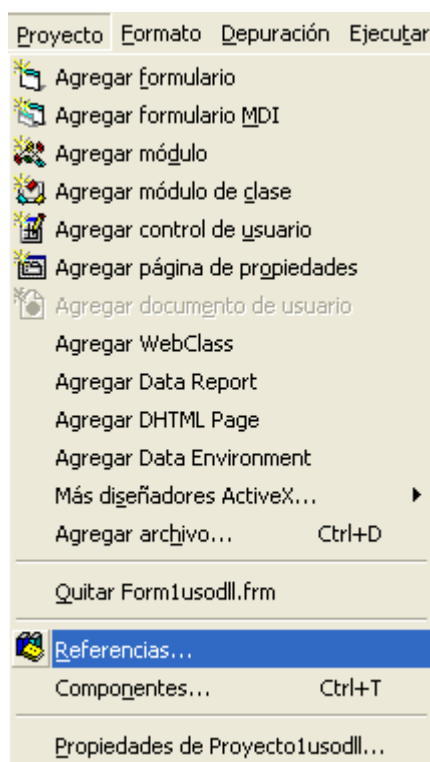
Dentro de Class1 se deben crear las funciones que se invocarán desde un ejecutable.

La clase debe tener las opciones de:

La propiedad Instancing de Class1 a 6 GlobalMultiUse, lo que sirve para no tener que definir previamente una variable para tener que usar esta Clase.

Para utilizarse en un proyecto ejecutable:

En el menú ‘Proyecto > Referencias’, se elige el proyecto creado anteriormente como dll (se refiere al nombre de proyecto registrado, no al nombre de la dll).



Para utilizarse esta dll en otros equipos donde no se compiló, deberá registrarse la dll regsvr32, para lo cual se deben seguir las especificaciones para este propósito que se tratan en el apartado de este trabajo llamado ‘**Registro manual de una dll, un ActiveX o un ocx en Windows**’.

## SECCIONES DE UN ARCHIVO DE DEFINICIÓN

Un ejemplo de fichero de definición .DEF podría ser:

```
NAME      HOLA
DESCRIPTION  'C++ Windows Hola Mundo'
EXETYPE   WINDOWS
CODE      MOVEABLE
DATA      MOVEABLE MULTIPLE
HEAPSIZE  1024
STACKSIZE 5120
EXPORTS   MainWindowProc
```

En la estructura de este tipo de archivos se manejan varias secciones para declaraciones e inicialización de procedimientos o definiciones necesarias para el funcionamiento del programa, etc.

### Secciones:

**NAME** Indica el nombre de un ejecutable normal. Si se desea construir una DLL, debe utilizarse la sección **LIBRARY** en sustitución de esta. Cualquier fichero DEF debe tener alguna de estas dos secciones, pero no ambas al mismo tiempo y el nombre debe coincidir con el del ejecutable o el de la dll, según sea el caso.

**DESCRIPTION** Cadena de información sobre las características del ejecutable o librería.

**EXETYPE** es **WINDOWS**.- La documentación que acompaña al compilador Borland C++ 5.5 indica que esta versión solo soporta **WINDOWS** como parámetro de la variable **EXETYPE**. No obstante, esta se mantiene solo por cuestiones de compatibilidad, y debe ser sustituida por **NAME** o **LIBRARY**.

**CODE** Relación directa con atributos por defecto del segmento de código. El argumento **MOVEABLE** significa que este segmento puede ser movido en tiempo de ejecución.

**DATA** Argumentos por defecto del segmento de datos. En este caso, **MOVEABLE** significa que puede ser movido en memoria en tiempo de ejecución. Por su parte, **MULTIPLE** garantiza que cada instancia de la aplicación dispondrá de su propio segmento de datos.

**HEAPSIZE** Tamaño del montón.

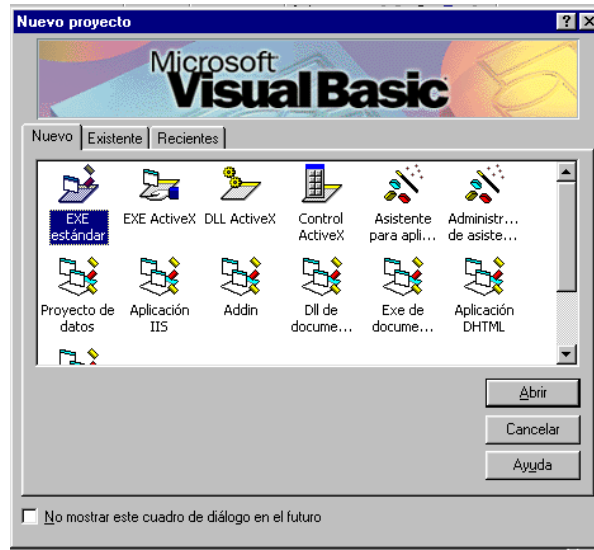
**STACKSIZE** Tamaño de la pila. No debe utilizarse esta variable si se pretende crear una DLL.

**EXPORTS** relaciona aquellas funciones de la aplicación HOLA que pueden ser invocadas por otras aplicaciones o por el mismo Windows (las "callbacks"). Los compiladores BC++ y MSVC proveen de los especificadores `__declspec(dllexport)` y `__export`, de tal forma que las funciones declaradas con ellos no necesitan ser incluidas en esta sección.

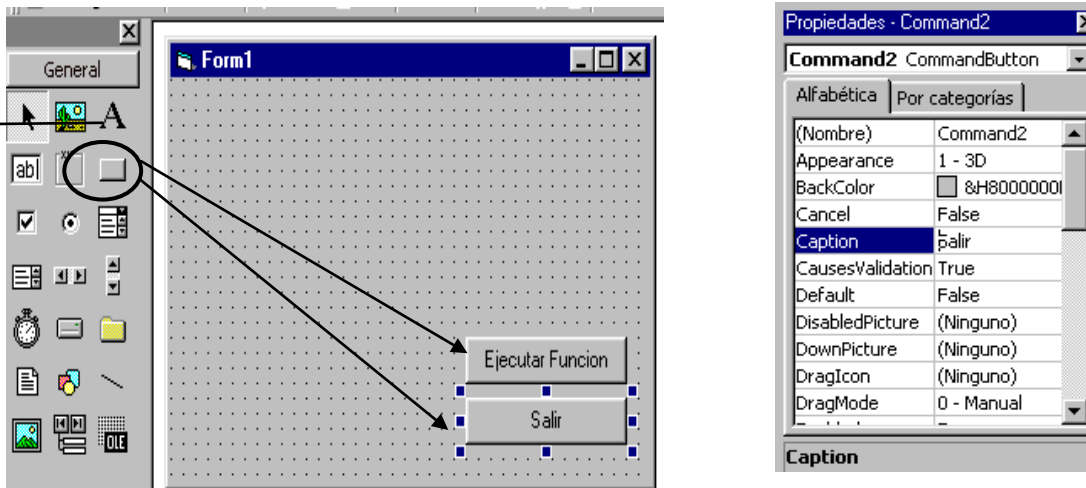
En este ejemplo no se incluye la sección **IMPORTS**, porque se supone que las únicas funciones que utiliza de otros módulos son de la API de Windows, y estas funciones son importadas mediante la inclusión de **IMPORT32.LIB** (esta librería es incluida automáticamente por BC++ en el proceso de construcción de un ejecutable Windows). Cuando una aplicación necesita invocar funciones externas (situadas en librerías dinámicas), estas funciones deben ser incluidas en la sección **IMPORTS**, o en una librería de importación enlazada estáticamente con la aplicación.

# LLAMADAS A FUNCIONES DE LA DLL DESDE VISUAL BASIC

1. Abrir un nuevo proyecto y seleccionar la opción de Exe Estandar

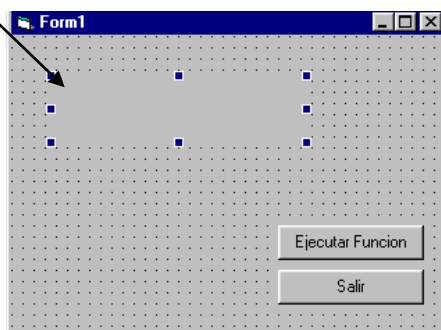


Insertar dos Botones en la forma inicial:



En la propiedad Caption cambiar el texto del Botón como se muestra en la figura anterior.

Insertar una etiqueta en la forma:

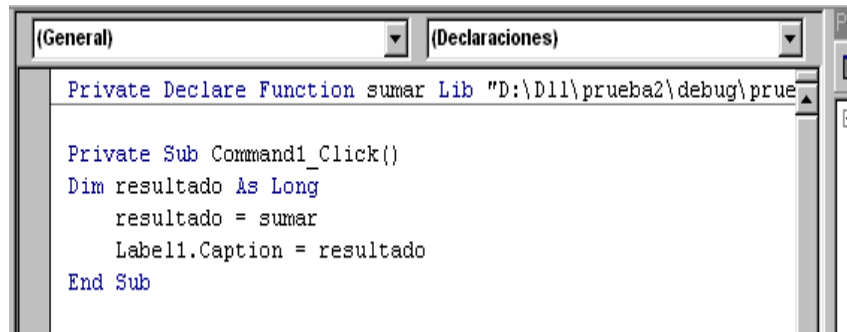


Hacer click en la forma, cambiar el texto de “Form1” en la propiedad “Caption” por “Uso de funciones externas con C”

Cambiar icono de la forma en propiedad “icon” por uno de su preferencia.

Hacer doble click en el botón “Ejecutar Función” y escribir el siguiente código.

Private Declare Function sumar Lib "D:\Dll\prueba2\debug\prueba2.dll" () As Long



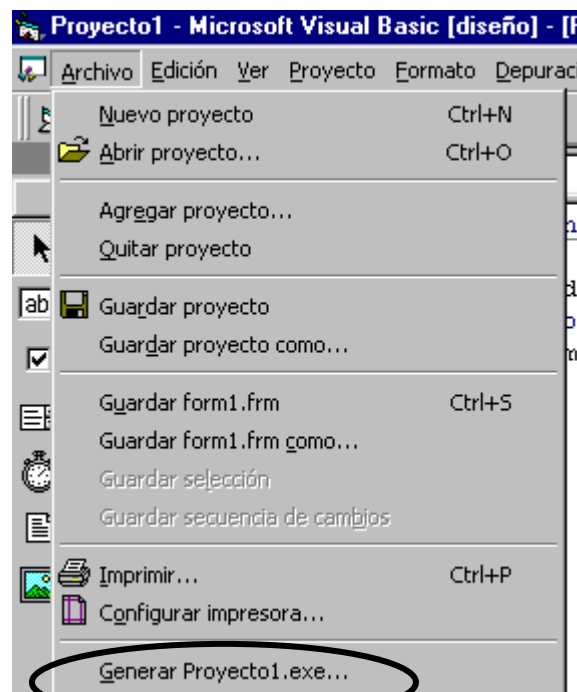
```
Private Declare Function sumar Lib "D:\Dll\prueba2\debug\prueba2.dll" () As Long

Private Sub Command1_Click()
    Dim resultado As Long
    resultado = sumar
    Label1.Caption = resultado
End Sub
```

Ejecutar el proyecto:



Construir ejecutable del proyecto:



## PASAR VALORES POR REFERENCIA HACIA LA FUNCIÓN DE LA DLL QUE SE EJECUTARÁ

- En Visual C++, en la declaración de la clase modificar (prueba2.h)  
Long sumar( void )  
por  
Long sumar( int &m )
- En la implementación de la función sumar (prueba2.cpp), modificar:  
long CPrueba2App::sumar( void) {  
por  
long CPrueba2App::sumar(int &m) {  
y  
return resultado;  
por  
return resultado + m;

- En Visual Basic, en la declaración de la función cambiar los argumentos de  
Private Declare Function sumar Lib "D:\Dll\prueba2\debug\prueba2.dll" () As Long  
a

```
Private Declare Function sumar Lib "D:\Dll\prueba2\debug\prueba2.dll" (ByRef m As Long) As Long
```

Con lo cual enviamos el valor por referencia (dirección de la variable) a la función de la DLL.

Agregar al código del procedimiento command1\_click

```
En  
Dim resultado As Long  
poner  
Dim resultado, m As Long
```

```
Agregar  
m = 50
```

y modificar

```
resultado = sumar ( m )
```

## PASO DE PARÁMETROS USANDO CADENAS DE CARACTERES

Se sigue el mismo procedimiento para crear una DLL tipo *MFC AppWizard (dll)* con la única modificación de las funciones que se agregan en la declaración de la clase y su desarrollo en el archivo tipo *cpp*. De igual forma agregaremos una extensión externa cuando compilemos por primera vez la *dll*, misma que contendrá el código en lenguaje C que se incluirá:

Esto es igual que una receta, hay pasos que se deben seguir tal como se indica para obtener el resultado correcto.:

- a) Primero se debe crear el archivo con terminación “.h” o “.c” o “.cpp”

En el cual se incluirá el siguiente código:

```
char *cadena(void) {
    char *cade = "Mensaje 1";
    return cade;
}

char *cadena2(void) {
    char *cade2 = "Mensaje 2";
    return cade2;
}
```

- b) Crear un proyecto tipo *MFC AppWizard (dll)* con Visual C++ y declarar función *switch1* en la clase del proyecto que se encuentra en el archivo “.h” donde la función derivada de la clase también se declara de tipo apuntador.
- c) Inmediatamente después se localiza el archivo “.cpp”, se abre y se localiza alguna región libre para escribir código que se encuentre después del constructor de la clase. En este ejemplo incluir:

```
char *CPruebaDllApp::switch1(char *m) {
    char *salida;
    char *temp;
    salida = (char *) malloc(sizeof(char));
    if(strcmp(m,"AA")==0) {
        temp = cadena();
        strcpy(salida, temp);
    }
    else if (strcmp(m,"EA")==0) {
        temp = cadena2();
        strcpy(salida, temp);
    }
}
```



```
        return(salida);
    }
}
```

- d) De igual forma, en el inicio del archivo “.cpp”, se debe incluir la siguiente definición:

```
#include "stdafx.h"
#include "PruebaDll.h"
//declaracion de funciones de c, codigo adicional ;
extern "C" {
#include "D:\dll\funcionc.h"
} // hasta aqui codigo adicional mio
```

- e) Por último, en el archivo con terminación “.def” se incluye el nombre de la o las funciones que se van a exportar en esta dll.

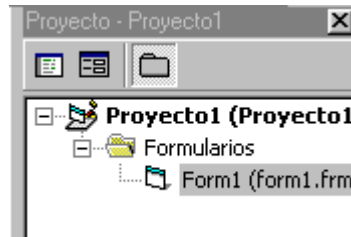
```
; PruebaDll.def : Declares the module parameters
LIBRARY      "PruebaDll"
DESCRIPTION  'PruebaDll Windows Dynamic Link Library'

EXPORTS
; Explicit exports can go here
DllCanUnloadNow PRIVATE
DllGetClassObject PRIVATE
DllRegisterServer PRIVATE
|
switch1
```

## Código de Visual Basic para verificar como funciona la función “switch”

Abrir un proyecto nuevo tipo Exe estandar y como código de la Form1 incluir lo siguiente:

Punto de entrada de la dll (declaración de la dll en visual basic)



Donde tenemos solamente una forma con dos botones ‘Command1’ y ‘Command2’:

```
Command1 Click
Option Explicit
Private Declare Function switch1 Lib _
    "c:\aaa\pruebadll\debug\pruebadll.dll" _
    (ByVal m As String) As String

Private Sub Command1_Click()
    Dim cadena, entrada As String
    entrada = CStr("AA")
    Label2.Caption = entrada
    cadena = switch1(entrada)
    Label1.Caption = cadena
End Sub

Private Sub Command2_Click()
    Dim cadena, entrada As String
    entrada = CStr("EA")
    Label2.Caption = entrada
    cadena = switch1(entrada)
    Label1.Caption = cadena
End Sub
```

## CONCLUSIONES

El continuo y rápido avance que se presenta en la Tecnología de Computación, y la necesidad de optimizar los recursos de software y de hardware, obligan a buscar alternativas de solución que permitan elevar la calidad y eficiencia de los productos de software que se desarrollan.

Por otra parte, tampoco se puede negar que el desarrollo de sistemas para ambiente Windows sigue y seguirá siendo por mucho tiempo una de las alternativas más utilizadas que se usará como plataforma para las aplicaciones que se utilizan en la resolución de problemas reales de diversos tipos.

Siendo Ensamblador y C lenguajes de bajo y medio nivel de máquina respectivamente, el grado de complejidad para programar con ellos es más alto que el de los lenguajes de alto nivel, motivo por lo cual resulta interesante contar con la posibilidad de poder utilizar ambos recursos, incluyendo en las aplicaciones creadas, solamente las rutinas necesarias, utilizando para ello el esquema de una Librería de Enlace Dinámico (Dll), a fin de que posteriormente, se puedan manipular por una o más aplicaciones, teniendo la gran ventaja de que éstos módulos sólo se cargarán en memoria cuando así sea requerido y se descargarán una vez que hayan cumplido con su objetivo

En este trabajo se presentan alternativas de “pegar” o “ligar” módulos llamados “dll” a fin de optimizar los recursos de memoria y aumentar el grado de eficiencia en la ejecución de las aplicaciones que las utilizan.

Otra gran ventaja es que existen varios compiladores y métodos para generar Dll's, de tal forma que es el diseñador y programador de un sistema, quien podrá decidir la vía de desarrollo más recomendable para la resolver un problema aplicando este tipo de tecnología computacional.