FUNCIONES

Los Funciones en c++ permiten modularizar sus programas. Todas las variables declaradas en las definiciones de Funciones son variables locales es decir solo se conocen en la función que se definen.

Casi todos las Funciones tienen una lista de parámetros que permiten comunicar información entre funciones. Los parámetros de una función también son variables locales.

La sintaxis para declarar un función es la siguiente :

```
tipo_de_valor_devuelto nombre_de_la_función ( lista_de_parámetros)
{
  Cuerpo de la funcion
}
```

Donde :

tipo_de_valor_devuelto : Indica el tipo de valor que se devuelve a la función invocadora. Si una función no devuelve un valor, el tipo_de_valor_devuelto se declara como void.

nombre_del_función: Es cualquier identificador válido

lista_de_parámetros: Es una lista separada por comas que contiene las declaraciones de las variables que se pasarán a la función. Si una función no recibe parámetros las lista_de_parámetros se deja vacía.

El cuerpo de la función: Es el conjunto de declaraciones e instrucciones que constituyen la función.

Si la función no devuelve valor, el control se devuelve al llegar a la llave derecha que termina la función o ejecutando la sentencia return;

Si la función devuelve un valor, la sentencia:

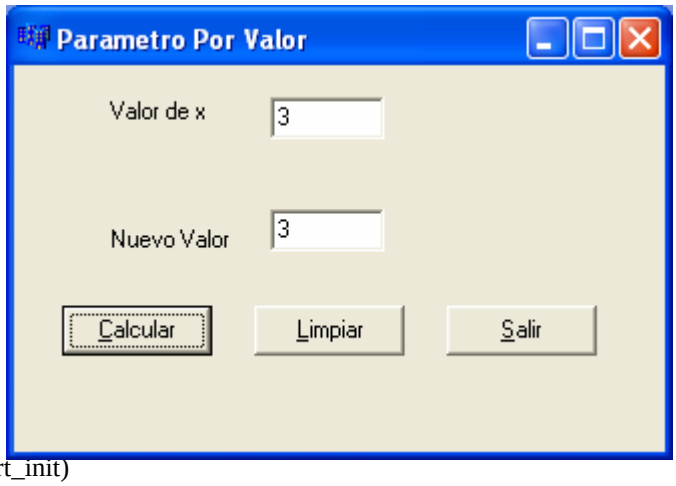
```
return expresión;
```

devuelve el valor de expresión.

Todas las variables declaradas en una función son variables locales, estas variables son reconocidas en la función en la cual fueron definidas. Los parámetros de una función son también variables locales de ésta.

Tipos de Parámetros de las Funciones

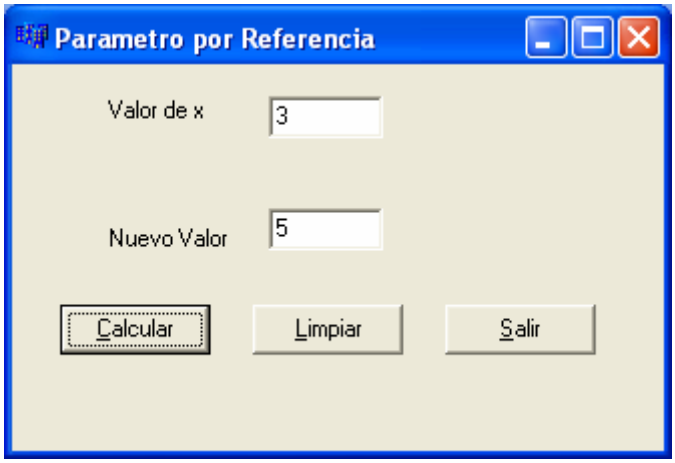
1) **Parámetros por Valor**: Son aquellos a través de los cuales se pasan valores a la función, es decir se hace una copia de la variable pasada como argumento. A estos parámetros se les conoce como parámetros de entrada.




```
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
//---------------------------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------------------------
void calculo(float x)
{
  x=x+2;
}

void __fastcall TForm1::btnCalcularClick(TObject *Sender)
{
   float x;
   x=EdX->Text.ToDouble();
   calculo(x);
   EdV->Text=x;
}
//---------------------------------------------------------------------------

void __fastcall TForm1::btnLimpiarClick(TObject *Sender)
{
  EdX->Clear();
  EdV->Clear();
  EdX->SetFocus();
}

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

La variable x del programa principal al llamar a la función calculo hace una copia de su valor al parámetro de la función, luego la que se incrementa es la variable x de la función y no la variable x del programa principal.

2) **Parámetros por referencia:** Son aquellos a través de los cuales se pasan referencias de las variables esto permite que las variables pasadas como argumento se puedan modificar.
Para declarar un parámetro por referencia se utiliza el operador referencia &.

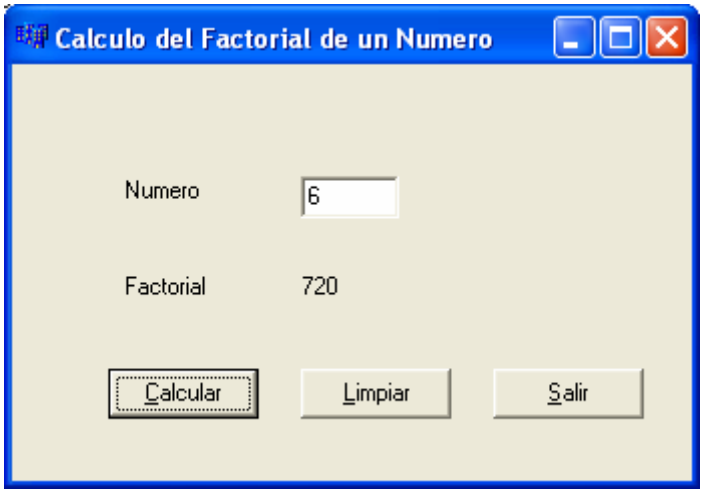



```
//---------------------------------------------------------------------------

void calculo(float &x)
{
  x=x+2;
}

void __fastcall TForm1::btnCalcularClick(TObject *Sender)
{
   float x;
   x=EdX->Text.ToDouble();
   calculo(x);
   EdV->Text=x;
}
//---------------------------------------------------------------------------



void __fastcall TForm1::btnLimpiarClick(TObject *Sender)
{
  EdX->Clear();
  EdV->Clear();
  EdX->SetFocus();
}
//---------------------------------------------------------------------------

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

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

La variable x del programa principal al llamar a la función calculo pasa la referencia de la variable esto hace que la variable x se pueda modificar.

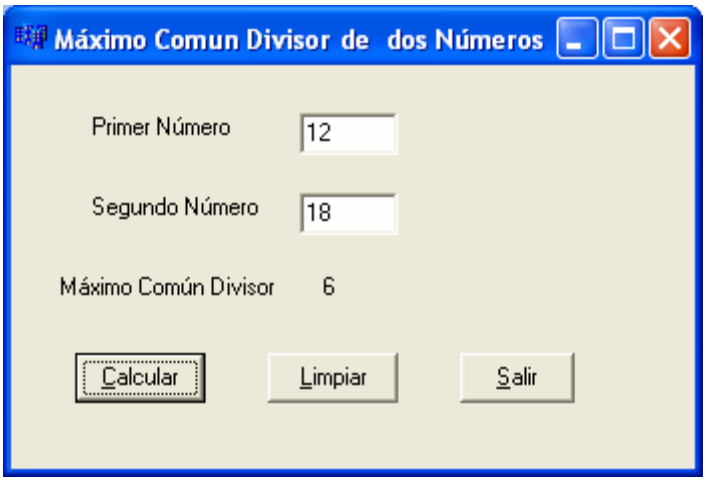
Ejercicios

1) Hacer un programa para calcular el factorial de un número entero n.



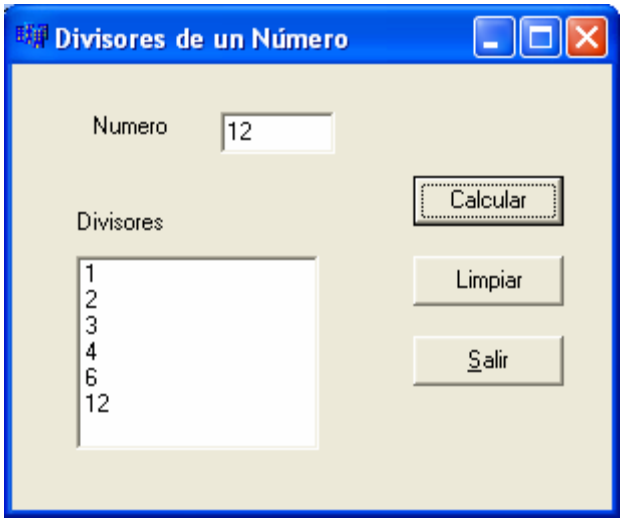

```
#include <vcl.h>
#pragma hdrstop
#include "Unit1.h"
#pragma package(smart_init)
#pragma resource "*.dfm"

TForm1 *Form1;
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{
}

int  factorial(int n)
{
  int f=1,i;
  for(i=1;i<=n;i++)
    f=f*i;
  return f;
}

void __fastcall TForm1::btnCalcularClick(TObject *Sender)
{
  int n;
  n=edN->Text.ToInt();
  lblF->Caption=factorial(n);

}


void __fastcall TForm1::btnLimpiarClick(TObject *Sender)
{
   edN->Clear();
   lblF->Caption="";
   edN->SetFocus();
}

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

2) Calcular el MCD de dos números enteros.




```
#include <vcl.h>
#pragma hdrstop
#include "Unit1.h"

#pragma package(smart_init)
#pragma resource "*.dfm"

TForm1 *Form1;

__fastcall TForm1::TForm1(TComponent*
Owner)
        : TForm(Owner)
{
}

int mcd(int x, int y)
{
  int d=2,p=1;
  while(d<=x && d<=y)
  {
     if(x % d == 0 && y % d ==0)
       {
         p=p*d;
         x=x/d;
         y=y/d;
       }
     else
         d++;
  }
  return p;
}

void __fastcall
TForm1::btnCalcularClick(TObject *Sender)
{
   int n1,n2;
   n1=edN1->Text.ToInt();
   n2=edN2->Text.ToInt();
   lblMcd->Caption=mcd(n1,n2);

}

void __fastcall
TForm1::btnLimpiarClick(TObject *Sender)
{
   edN1->Clear();
   edN2->Clear();
   lblMcd->Caption="";
   edN1->SetFocus();
}

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

3) Programa para reportar todos los divisores de un número entero N



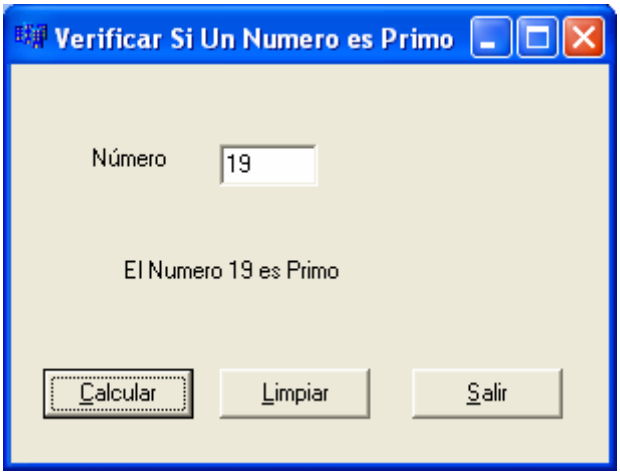

```
#include <vcl.h>
#pragma hdrstop

#include "Unit1.h"
#pragma package(smart_init)
#pragma resource "*.dfm"

TForm1 *Form1;
__fastcall TForm1::TForm1(TComponent* Owner) : TForm(Owner)
{
}

void reporteDivisores(int n,TListBox *lst)
{
  int i;
  lst->Items->Clear();
  for(i=1;i<=n;i++)
  {
    if(n % i ==0)
      lst->Items->Add(i);
  }
}

void __fastcall TForm1::btnCalcularClick(TObject *Sender)
{
   int n;
   n=edN->Text.ToInt();
   reporteDivisores(n,lstDivisores);
}

void __fastcall TForm1::btnLimpiarClick(TObject *Sender)
{
  edN->Clear();
  lstDivisores->Items->Clear();
  edN->SetFocus();
}

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

4) Programa para reportar los factores primos de un número entero n


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

#include "Unit1.h"
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;

__fastcall TForm1::TForm1(TComponent*
Owner)
        : TForm(Owner)
{
}

void factoresPrimos(int n,TListBox *lst)
{
  int d=2;
  lst->Items->Clear();
  while(n>1)
  {
    if(n % d ==0)
      {
       lst->Items->Add(d);
       n=n/d;
      }
    else
       d++;
  }
}

void __fastcall
TForm1::btnCalcularClick(TObject *Sender)
{
 int n;
 n=edN->Text.ToInt();
 factoresPrimos(n,lstFactores);
}
```

```
void __fastcall
TForm1::btnLimpiarClick(TObject *Sender)
{
  edN->Clear();
  lstFactores->Clear();
  edN->SetFocus();
}
void __fastcall TForm1::btnSalirClick(TObject
*Sender)
{
   Close();
}
```

5) Programa para ingresar un numero y se reporte si es primo

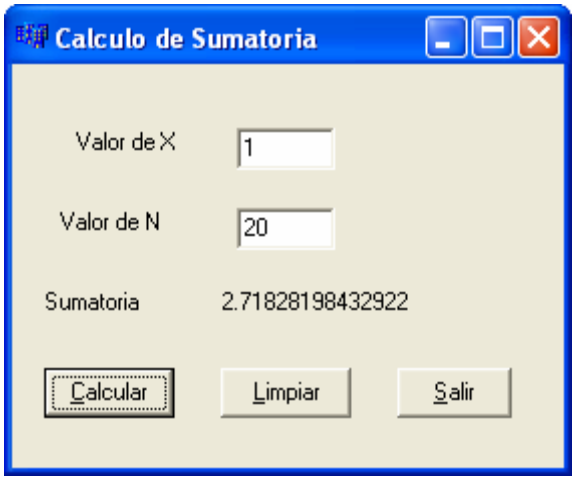



```
#include <vcl.h>
#pragma hdrstop
#include "Unit1.h"
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{
}

bool esPrimo(int n)
{
  int d=1;
  do{
    d=d+1;
  }while( n%d!=0 && d*d<=n);
  if(d*d>n) return true;
  else return false;
}

void __fastcall TForm1::btnCalcularClick(TObject *Sender)
{
  int n;
  n=edN->Text.ToInt();
  if(esPrimo(n))
    lblMensaje->Caption="El Numero "+IntToStr(n) +" es Primo";
  else
    lblMensaje->Caption="El Número "+IntToStr(n)+" No es Primo";
}

void __fastcall TForm1::btnLimpiarClick(TObject *Sender)
{
 edN->Clear();
 lblMensaje->Caption="";
 edN->SetFocus();
}

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

6) Calcular el valor de la siguiente sumatoria:

s = 1+ x + x2/2! + x3/3! + x4/4! + ... + xn/n!

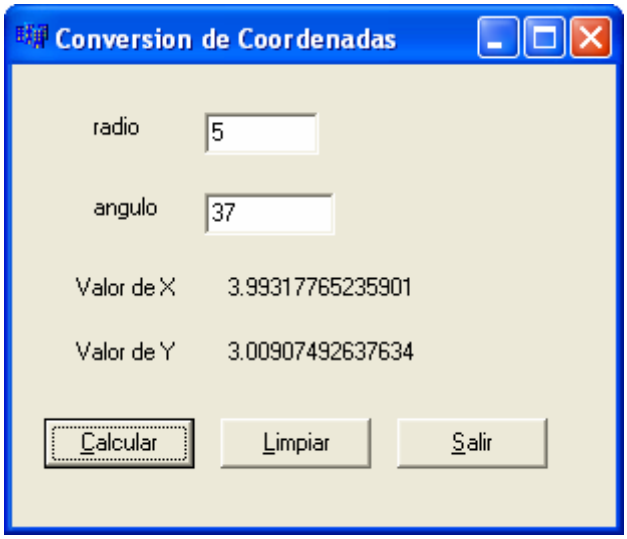
Se debe Ingresar el valor de x, y el valor de n>0.




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

#include "Unit1.h"

#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;

__fastcall TForm1::TForm1(TComponent*
Owner)
        : TForm(Owner)
{
}

float suma(float x,int n)
{
  float s=1,f=1,i,p=1;
  for(i=1;i<=n;i++)
  {
    f=f*i;
    p=p*x;
    s=s+p/f;
  }
  return s;
}

void __fastcall
TForm1::btnCalcularClick(TObject *Sender)
{
   float x;
   int n;
   x=edX->Text.ToDouble();
   n=edN->Text.ToInt();
   lblS->Caption=suma(x,n);
}
```

```
void __fastcall
TForm1::btnLimpiarClick(TObject *Sender)
{
   edX->Clear();
   edN->Clear();
   lblS->Caption="";
   edX->SetFocus();
}

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

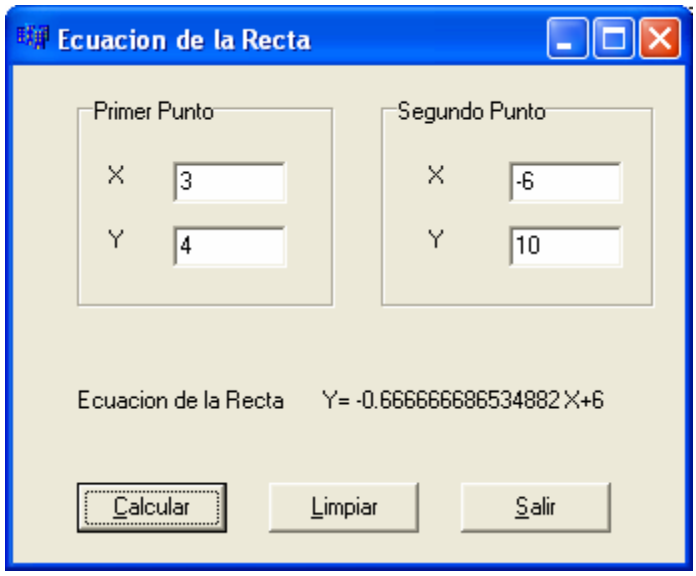
7) Programa para ingresar el valor de un Punto en coordenadas Polares y reporte su equivalente en coordenadas cartesianas.




```
#include <vcl.h>
#pragma hdrstop
#include "Unit1.h"
#include<math.h>

#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;

__fastcall TForm1::TForm1(TComponent*
Owner)   : TForm(Owner)
{
}
void calculo(float r, float t, float &x, float &y)
{
   // convertimos el angulo de sexagesimales a
radianes
   t=t*M_PI/180;
   x=r*cos(t);
   y=r*sin(t);
}

void __fastcall
TForm1::btnCalcularClick(TObject *Sender)
{
  float r,t,x,y;
  r=edR->Text.ToDouble();
  t=edA->Text.ToDouble();
  calculo(r,t,x,y);
  lblX->Caption=x;
  lblY->Caption=y;
}

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

```
void __fastcall
TForm1::btnLimpiarClick(TObject *Sender)
{
  edR->Clear();
  edA->Clear();
  lblX->Caption="";
  lblY->Caption="";
  edR->SetFocus();
}
```

3) Ingrese 2 puntos del plano cartesiano y reporte la ecuacion de la recta que los contiene.




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

#include "Unit1.h"
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;

__fastcall TForm1::TForm1(TComponent*
Owner)
      : TForm(Owner)
{
}

void calculo(float x1, float y1, float x2, float y2,
float  &m, float &b)
{
  m=(y2-y1)/(x2-x1);
  b=y1-m*x1;
}

void __fastcall
TForm1::btnCalcularClick(TObject *Sender)
{
  float x1,y1,x2,y2,m,b;

  x1=edX1->Text.ToDouble();
  y1=edY1->Text.ToDouble();
  x2=edX2->Text.ToDouble();
  y2=edY2->Text.ToDouble();
  calculo(x1,y1,x2,y2,m,b);
  lblEcuacion->Caption="Y=
"+FloatToStr(m)+" X";
  if (b>0)
    lblEcuacion->Caption=lblEcuacion-
>Caption+"+"+FloatToStr(b);
  else
    lblEcuacion->Caption=lblEcuacion-
>Caption+" "+FloatToStr(b);
}

void __fastcall
TForm1::btnLimpiarClick(TObject *Sender)
{
  edX1->Clear();
  edY1->Clear();
  edX2->Clear();
  edY2->Clear();
  lblEcuacion->Caption="";
  edX1->SetFocus();
}

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

Sobrecarga de funciones

C++ permite crear más de una función con el mismo nombre las cuales se diferenciarán por el número de parámetros utilizados ó en el tipo de éstos o en ámbos. El valor regresado por las funciones puede ser igual ó diferente sin embargo dos o más funciones con el mismo nombre y parámetros pero con valor de retorno diferente generará un mensaje de error por parte del compilador. Al proceso de llamar a más de una función con el mismo nombre se le llama sobrecarga de funciones.

Ejemplo 1:

```
int abs(int numero)
{
    if(numero<0) return –numero;
    else return numero;
}

long abs(long numero)
{
    if(numero<0) return –numero;
    else return numero;
}

double abs(double numero)
{
    if(numero<0) return –numero;
    else return numero;
}
```

- Ejemplo 2 :

```
int mayor(int a, int b)
{
  if(a > b) return a;
  else return b;

}

char mayor(char a, char b)
{
   if(a > b) return a;
   else return b;
}

double mayor(double a, double b)
{
   if(a > b) return a;
   else return b;
}
```

- Ejemplo 3:

```
int mayor(int a, int b)
{
   if(a > b) return a;
   else return b;

}

int mayor(int a, int b, int c)
{
  return mayor(mayor(a, b), c);
}
```

```
int mayor(int a, int b, int c, int d)
{
return mayor(mayor(a, b), mayor(c, d));
}
```

Argumentos por defecto

- C++ permite tener valores por defecto para los parámetros. Esto supone que, si no se pasa el parámetro correspondiente, se asume un valor predefinido.

Ejemplo:

```
void calculo (int  t = 2)
```

En este caso, estamos definiendo un valor, 2, que tomará la variable t en caso de que no se pase nada en la llamada a la función.

Si se llama de la siguiente manera : calculo (); Aunque no se ha pasado ningún argumento, la función trabajará. Como no se ha pasado ningún valor, t toma el valor 2, y la función se ejecuta.

Si se le llama con calculo(456) ahora t toma el valor de 456.

- La gramática de C++ exige que los parámetros con valores por defecto deben ser los últimos en la lista de parámetros, y que si en una ocasión falta algún argumento, los que le siguen también deben faltar (adoptar también los valores por defecto).

Ejemplo:

```
void funcion2 (int var1, int var2, int var3 = 0, int var4 = 0)
```

es la forma correcta de hacerlo.

En cambio:

```
void funcion2 (int var1, int var2 = 0, int var3 = 0, int var4)
```

no es correcto.

La razón es que si quisiéramos llamar a esta función con tres argumentos, el compilador no distinguiría si los valores que pasamos corresponden a var1, var2 y var3 o a var1, var3 y var4.

Resumiendo, hemos de seguir las siguientes reglas al declarar funciones con parámetros por defecto:

1) Una vez que a uno de los parámetros se le asigna un valor por defecto, los parámetros que le siguen en la lista de argumentos deben tener también un valor por defecto. No se puede dejar un hueco en medio de la lista.

2) Los valores por defecto deben ser del tipo correcto (de un tipo compatible con el declarado) o el compilador dará error.

3) Los valores por defecto pueden ser declarados en el prototipo o en la definición de la función, pero no en ambos. Como cuestión de estilo, se recomienda que los valores por defecto se declaren en el prototipo.

Ejemplo:

```
int volumen (int longitud, int anchura = 2, int altura = 3);
{
  return longitud * anchura * altura;
}
```

En el boton Calcular puedo llamar de la siguiente manera

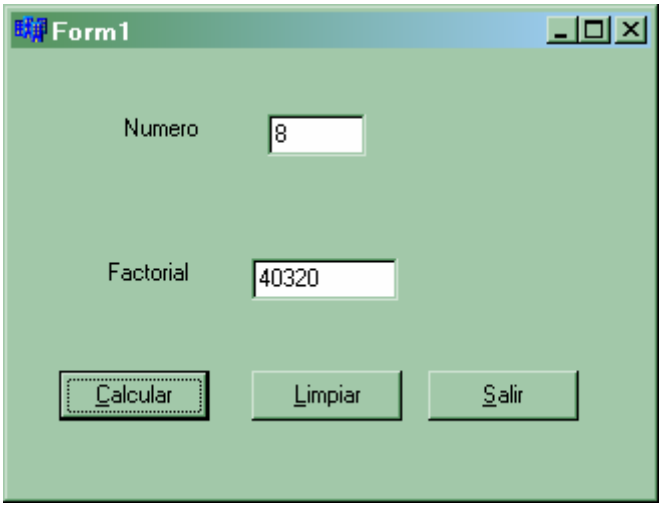
```
 int x = 10, y = 12, z = 15,a,b,c,d;
a = volumen(x,y,z); // a toma el valor de  1800
b = volumen(x,y);  // b toma el valor de 360
c = volumen(x);   // c toma el valor de 60
d=volumen(x,7); // d toma el valor de  210
e=volumen(5,5,5);  // e toma el valor de  125
float calculo (float x, float y = 0);
```

Recursividad

En programación, una función es recursiva si en el ámbito de esa función hay una llamada a sí misma.
Para decidir hacer un programa recursivo se deben de tener al menos dos cosas muy claras:
1. El paso base: Esta es la clave para terminar la recursión, es cuando deja de hacer llamadas a la función recursiva y hace evaluaciones devolviendo los resultados. Además se debe asegurar de que es posible entrar a este paso.
2. El paso recursivo: Es la parte de la definición que hace llamadas a esa misma función y que es la causante de las inserciones en la pila, almacenando en cada una de las llamadas, información del programa, del estado de sus variables locales y globales.
Frecuentemente tanto el paso base como el paso recursivo, se encuentran en una sentencia condicional if, pero porsupuesto que es posible usar cualquier otra sentencia de control, dependiendo de las necesidades particulares del problema.

Ejercicios

1) Programa para calcular el factorial de un número.




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

#include <vcl.h>
#pragma hdrstop

#include "Unit1.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
//---------------------------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------------------------
```

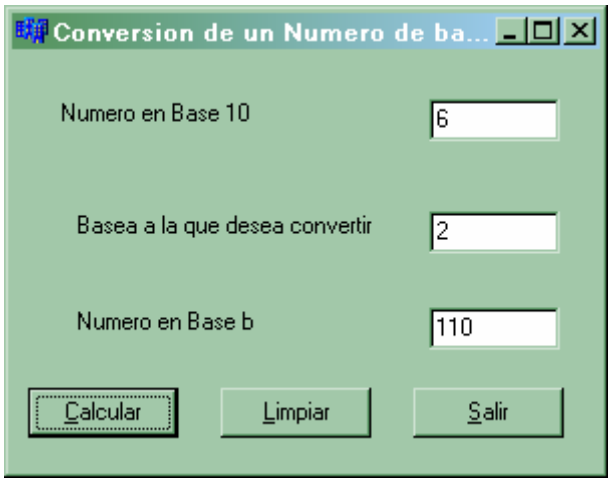
```
int factorial(int n)
{
  if(n==0) return 1;
  else return n*factorial(n-1);
}


void __fastcall TForm1::btnCalcularClick(TObject *Sender)
{
  int num;
  num=edN->Text.ToInt();
  edF->Text=factorial(num);
}
//---------------------------------------------------------------------------

void __fastcall TForm1::btnLimpiarClick(TObject *Sender)
{
  edN->Clear();
  edF->Clear();
  edN->SetFocus();

}
//---------------------------------------------------------------------------
void __fastcall TForm1::btnSalirClick(TObject *Sender)
{
  Close();

}
//---------------------------------------------------------------------------
```

2) Programa para reportar un numero al revés.


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

#include <vcl.h>
```

```
#pragma hdrstop

#include "Unit1.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
//---------------------------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------------------------

String reves(int N)
{
  if(N==0) return "";
  else return IntToStr(N%10)+reves(N/10);
}

void __fastcall TForm1::btnCalcularClick(TObject *Sender)
{
  int N;
  N=edN->Text.ToInt();
  edR->Text=reves(N);
}
//---------------------------------------------------------------------------

void __fastcall TForm1::btnLimpiarClick(TObject *Sender)
{
 edN->Clear();
 edR->Clear();
 edN->SetFocus();
}
//---------------------------------------------------------------------------
void __fastcall TForm1::btnSalirClick(TObject *Sender)
{
  Close();
}
//---------------------------------------------------------------------------
```

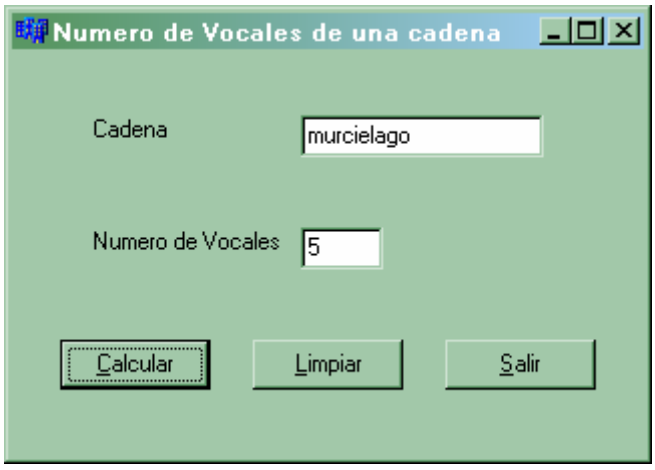
3) Programa para convertir un numero de base 10 a base b (entre 2 y 9)




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

#include <vcl.h>
#pragma hdrstop

#include "Unit1.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
//---------------------------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------------------------

String conversionBase(int n, int b)
{
  if(n<b)
      return IntToStr(n);
  else
      return conversionBase(n/b,b)+IntToStr(n%b);
}


void __fastcall TForm1::btnCalcularClick(TObject *Sender)
{
  int n,b;
  n=edN->Text.ToInt();
  b=edB->Text.ToInt();
  edR->Text=conversionBase(n,b);
```

```
}
//---------------------------------------------------------------------------


void __fastcall TForm1::btnLimpiarClick(TObject *Sender)
{
  edN->Clear();
  edB->Clear();
  edR->Clear();
  edN->SetFocus();
}
//---------------------------------------------------------------------------
void __fastcall TForm1::btnSalirClick(TObject *Sender)
{
  Close();
}
//---------------------------------------------------------------------------
```

4) Cuente recursivamente la cantidad de vocales que tiene una cadena.




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

#include <vcl.h>
#pragma hdrstop

#include "Unit1.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;

bool esVocal(char x)
{
```

```
  if(x=='a' || x=='e' || x=='i' || x=='o' || x=='u')
      return true;
  else
      return false;
}

int numVocales(String cadena, int n)
{
   if(n==0) return 0;
   else
       if(esVocal(cadena[n])) return 1+numVocales(cadena,n-1);
       else return numVocales(cadena,n-1);
}

//---------------------------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
       : TForm(Owner)
{
}
//---------------------------------------------------------------------------

void __fastcall TForm1::btnCalcularClick(TObject *Sender)
{
    edNV->Text=numVocales(edCadena->Text,edCadena->Text.Length());
}
//---------------------------------------------------------------------------

void __fastcall TForm1::btnLimpiarClick(TObject *Sender)
{
   edCadena->Clear();
   edNV->Clear();
   edCadena->SetFocus();
}
//---------------------------------------------------------------------------
void __fastcall TForm1::btnSalirClick(TObject *Sender)
{
   Close();

}
//---------------------------------------------------------------------------
```

Ejercicios propuestos

1.- Calcule el cociente y el residuo que resultan de dividir dos números enteros.
2.- Calcule la suma de los digitos de un numero.

3.- La relación recurrente

$$x^n = \begin{cases} 1 & si\ n = 0 \\ x * x^{n-1} & si\ n > 0 \\ x^{n+1} / x & si\ n < 0 \end{cases}$$

define x como la potencia n-ésima de todos los enteros. Escribir un procedimiento recursivo llamado `Potencia` que calcule x a la n-ésima potencia para un entero x.

4.- El máximo común divisor de dos enteros positivos M y N se puede definir como:

$$MCD(M,N) = \begin{cases} N & si\ N \le M\ y\ M \bmod N = 0 \\ MCD(N,M) & si\ M < N \\ MCD(N, M \bmod N) & en\ otro\ caso \end{cases}$$

Escribir una función recursiva que permita calcular el máximo común divisor de dos enteros positivos.

7.- En general, el N-ésimo término de la sucesión de Fibonacci, se define como:

$$F(A,B,N) = \begin{cases} A & si\ N = 1 \\ B & si\ N = 2 \\ F(B, A+B, N-1) & si\ N > 2 \end{cases}$$

Donde A y B son los números que originan la secuencia. Escribir un programa que compute el N-ésimo término de la sucesión de Fibonacci usando diferentes números para originar la secuencia.

8.- La función Q se define como:

$$Q(N) = \begin{cases} 1 & si\ N \le 2 \\ Q(N - Q(N-1)) & si\ N > 2 \end{cases}$$

Escribir dos procedimientos, uno recursivo y otro no recursivo, para computar el valor de esta función para varios valores de N. ¿Puedes escribir una función que haga uso de más memoria para así aumentar la velocidad de los cálculos recursivos?.

9.- El valor del N-ésimo término del polinomio de Legendre se puede calcular usando las siguientes fórmulas:

$P_0(x) = 1$
$P_1(x) = x$

$$P_N(x) = \frac{2*N-1}{N} * x * P_{N-1}(x) - \frac{N-1}{N} * P_{N-2}(x)$$

Escribir una función recursiva que compute el N-ésimo término del polinomio de Legendre.

10.- Escribir una función recursiva para computar la función de Ackermann, la cual se define como:

$$A(M,N) = \begin{cases} N+1 & \text{si } M = 0 \\ A(M-1,N) & \text{si } M \neq 0 \text{ and } N = 0 \\ A(M-1,A(M,N-1)) & \text{si } M \neq 0 \text{ and } N \neq 0 \end{cases}$$

donde M y N son números cardinales.

11.- Para cualesquiera enteros no negativos N y K, donde 0<=K<=N, el coeficiente binomial C(N,K) se define como:

$$C(N,K) = \frac{N!}{K!*(N-K)!}$$

Esta fórmula verifica que:

1. C(N,K) = C(N, N-K); es decir, la lista de coeficientes binomiales es simétrica con respecto a su valor central.
2. C(N,N) = 1 y por simetría C(N,0) = 1.
3. C(N,N-1)=N y por simetría C(N,1) = N.

El problema para computar C(N,K) haciendo uso de la fórmula anterior estriba en que N! crece muy rápidamente.Por lo que necesitamos un método alternativo para calcular dichos coeficientes. Para ello podemos hacer uso del hecho de que para cualquier K que satisfaga 1<=K<=N-1:

$$C(N,K) = \frac{C(N,K-1)*(N-K+1)}{K}$$

Escribir un programa que calcule el coeficiente binomial C(N,K) para varios N y K.

13.- En un cuadrado cuyo lado es a, se unen los puntos medios de sus 4 lados, formandose otro cuadrado cuyos puntos medios se unen tambien formando otro cuadrado, y asi sucesivamente. Calcular la suma de los perimetros de los n primeros cuadrados asi formados.

14.- Calcular el producto de los digitos de un numero

15.- Verificar si todos los digitos de un numero son iguales

16.- Verificar si los digitos de un numero estan en escalera ascendentemente

17.- Convertir un numero de base 10 a base b

18.- Calcule el m.c.m. de dos numeros