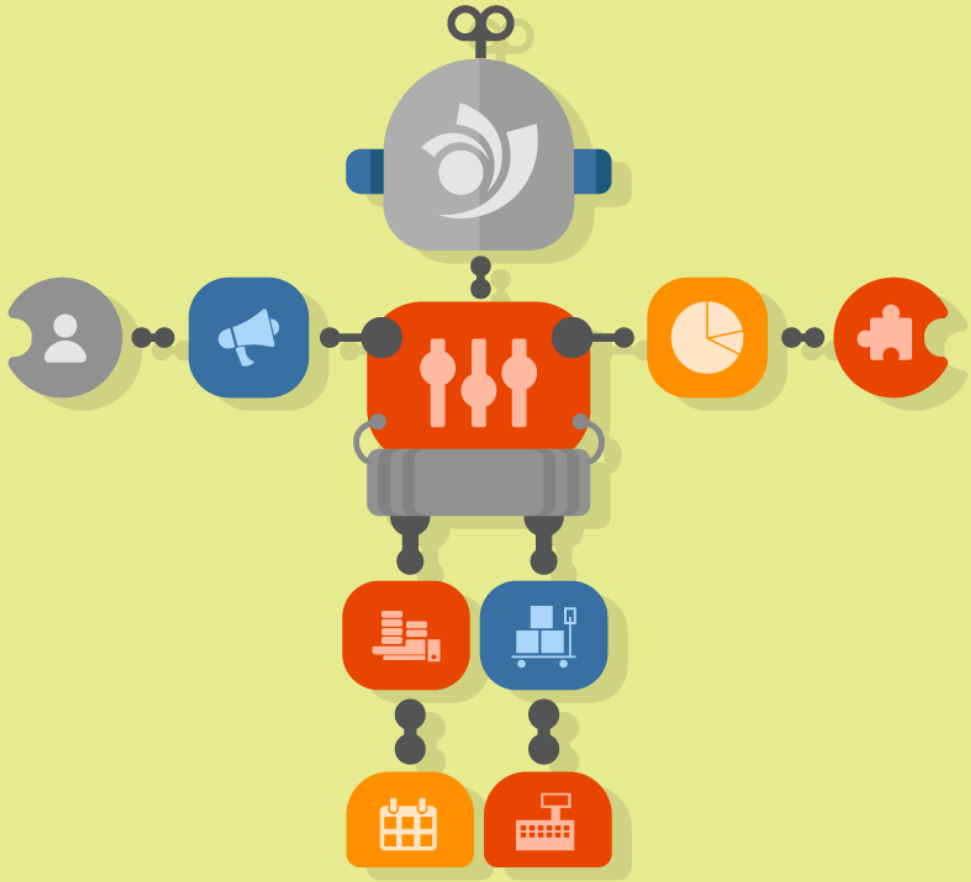
UNIVERSIDAD NACIONAL DE TRUJILLO

UNT

PROGRAMACIÓN MODULAR



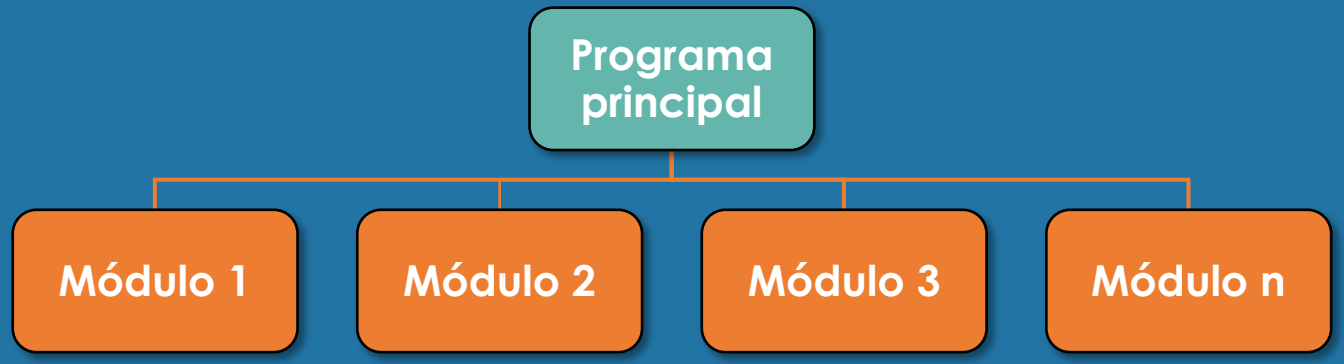
Mg. Marcelino Torres Villanueva

DEFINICIÓN

La programación modular es una técnica que consiste en dividir un programa en tareas y dar origen a la creación de pequeños programas llamados módulos, subprogramas o subrutinas.

Con la finalidad de simplificar la elaboración y mantenimiento del mismo, donde cada módulo se codifica y se procesa de manera independiente, sin importar los detalles de otros módulos.

En C++ a cada módulo o subprograma se le conoce como función; en este lenguaje se trabaja a base de funciones y, de manera general, los programas se elaboran combinando funciones que el programador escribe y funciones "predefinidas" disponibles en la biblioteca estándar de C.



Programa principal y funciones

La base de la programación en C++ es la función, ya que constituye una parte fundamental de la codificación en el proceso de solución de problemas. Un programa contiene una o más funciones en uno o más archivos.

Una de las funciones es main(), donde se inicia la ejecución del programa. El resto de las funciones se llaman desde main() y desde el interior de otras 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.

Prototipo de una función

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.

SINTAXIS

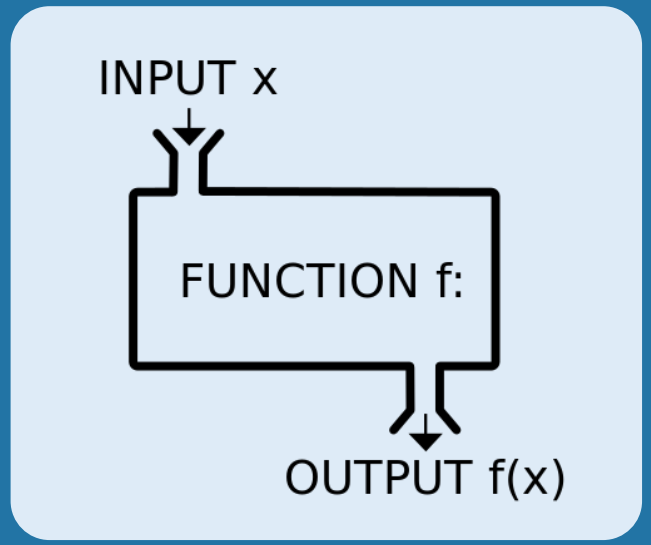
```
tipo_de_valor_devuelto  nombre_de_la_función ( lista_de_parámetros){
   Declaraciones e instrucciones
}
```

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.

Cuando un programa encuentra una llamada a una función, se transfiere el control desde el punto de invocación a la función invocada, se ejecuta la función y el control regresa a la función invocadora.



Una función invocada puede devolver el control a la invocadora de tres maneras:

- ✓ 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.

Pseudocódigo

```
Variables globales
Constantes
nombre_funcion_1(var1, varn)
nombre_funcion_n(var1, varn)
función principal()
    variables locales
    cuerpo del programa
    nombre_funcion_1(var1, varn)
    nombre_funcion_n(var1,varn)
fin función principal
```

```
nombre_funcion_1(var1, varn)
    Variables locales
    Instrucciones
fin función_1
nombre_funcion_n(var1, varn)
    Variables locales
    Instrucciones
fin función_n
```

Código

```
#include <iostream>
using namespace std;
//Variables globales
//Constantes
//tipo_de_valor_devuelto  nombre_de_la_función (lista_de_parámetros)
int nombre_funcion_1(tipo var1,tipo varn);
void nombre_funcion_n(tipo var1,tipo varn);
int main(int argc, char *argv[]) {
    //Variables locales
    //Cuerpo del programa
    //Llamado a las funciones
    nombre_funcion_1(tipo var1,tipo varn);
    nombre_funcion_n(tipo var1,tipo varn);
    return 0;
}
//Desarrollo de las funciones
int nombre_funcion_1(tipo var1,tipo varn){
    //Variables locales
    //Instrucciones
}

void nombre_funcion_n(tipo var1,tipo varn){
    //Variables locales
    //Instrucciones
}
```

VARIABLES LOCALES

EJERCICIOS RESUELTOS

Ejercicios resueltos

01. Desarrollar una aplicación que imprima un listado de los colores primarios, aplicando funciones.

	PSEUDOCODIGO
	rojo() imprimir " ROJO" **fin_rojo** **verde()** imprimir "VERDE" **fin_verde** **azul()** imprimir "AZUL" **fin_azul**

PSEUDOCODIGO

```
Algoritmo colores

imprimir "Los colores primarios son: "
rojo()
verde()
azul()

fin_algoritmo
```

CODIFICACION	SALIDA DE PANTALLA

`#include <iostream>`
`using namespace std;`
`void rojo();`
`void verde();`
`void azul();`
`int main(int argc, char *argv[]) {`
`  cout<<"Los colores primarios son: "<<endl;`
`  rojo();`
`  verde();`
`  azul();`
`  return 0;`
`}`
`void rojo(){`
`  cout<<"ROJO"<<endl;`
`}`
`void verde(){`
`  cout<<"VERDE"<<endl;`
`}`
`void azul(){`
`  cout<<"AZUL"<<endl;`
`}`
	Los colores primarios son:
	ROJO
	VERDE
	AZUL

Ejercicios resueltos

02. Desarrollar una aplicación para calcular la suma de dos números enteros. Use Funciones.

PSEUDOCODIGO

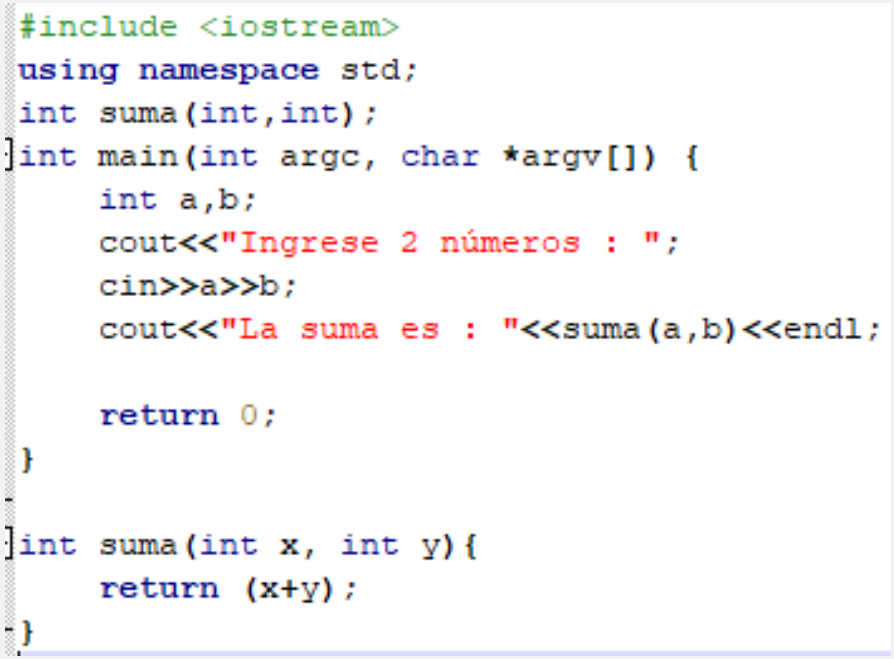
```
Algoritmo sumar_numeros

    entero a,b
   escribir "Ingrese 2 números: "
   leer a,b
   escribir "La suma es: ", suma(a,b)

fin_algoritmo



entero suma(entero x, entero y)
     retornar x+y
fin_suma
```

CODIFICACION	SALIDA DE PANTALLA
	Ingrese 2 números : 10 23 La suma es : 33

```cpp
#include <iostream>
using namespace std;
int suma(int,int);
int main(int argc, char *argv[]) {
    int a,b;
    cout<<"Ingrese 2 números : ";
    cin>>a>>b;
    cout<<"La suma es : "<<suma(a,b)<<endl;

    return 0;
}

int suma(int x, int y){
    return (x+y);
}
```

Ejercicios resueltos

03. Desarrollar una aplicación que calcule la factorial de un número entero. Use Funciones. Por ejemplo, el factorial de 5 sería: 5! = 1x2x3x4x5 = 120

PSEUDOCODIGO

```
real factorial (entero n)

    real f
    entero i
    f ← 1
    para i ← 1 hasta n inc 1 hacer
        f ← f * i
    fin_para
    retornar f

fin_factorial
```

PSEUDOCODIGO

Algoritmo factorial

```
entero n
Hacer
    escribir "Ingrese el número: "
    leer n
mientras n<=0
escribir "El factorial es: ", factorial(n)
```

fin_factorial

CODIFICACION	SALIDA DE PANTALLA

`#include <iostream>`
`using namespace std;`
`float factorial(int);`
`int main(int argc, char *argv[]) {`
`    int n;`
`    do{`
`        cout<<"Ingrese el número: ";`
`        cin>>n;`
`    }while(n<=0);`
`    cout<<"El factorial es: "<<factorial(n)<<endl;`
 	
`    return 0;`
`}`
`float factorial(int n){`
`    float f=1;`
`    int i;`
`    for(i=1;i<=n;i++)`
`        f=f*i;`
`    return f;`
`}`
	Ingrese el número: 5
	El factorial es: 120

Ejercicios resueltos

04. Desarrollar una aplicación para reportar todos los divisores de un número entero N que se ingrese por teclado. Use Funciones.

PSEUDOCODIGO

```
listaDivisores(entero n)

    entero i
    escribir "Los divisores de ", n, " son: "
    para i ← 1 hasta n inc 1 hacer
        si n mod i = 0 entonces
            escribir i
        fin_si
    fin_para

fin_listaDivisores
```

PSEUDOCODIGO

Algoritmo divisores

```
entero num
Hacer
    escribir "Ingrese el número: "
    leer num
mientras num<=0
listaDivisores(num)
```

fin_algoritmo

CODIFICACION	SALIDA DE PANTALLA

```cpp
#include <iostream>
using namespace std;
void listaDivisores(int);
int main(int argc, char *argv[]) {
    int num;
    do{
        cout<<"Ingrese el número: ";
        cin>>num;
    }while(num<=0);
    listaDivisores(num);
    return 0;
}
void listaDivisores(int n){
    int i;
    cout<<"Los divisores de "<<n<<" son: "<<endl;
    for(i=1;i<=n;i++)
    {
        if(n % i ==0)
            cout<<i<<" ";
    }
    cout<<endl;
}
```

Ingrese el número: 12

Los divisores de 12 son:

1 2 3 4 6 12

Ejercicios resueltos

05. Desarrollar una aplicación que permita ingresar un número entero positivo y reportar su tabla de multiplicar. Use Funciones.

PSEUDOCODIGO

```
entero leeNum()

    entero num
    Hacer
        escribir "Ingrese número: "
        leer num
    mientras num<=0
    retornar num

fin_leeNum
```

PSEUDOCODIGO

```
mostrarTabla(entero num)
    entero i
    escribir "Tabla de multiplicar del número",
    num
    para i ←1 hasta 12 inc 1 hacer
        escribir num, " * ", i , " = " , num * i
    fin_para
fin_mostrarTabla



Algoritmo tabla_de_multiplicar

    Declarar num como entero
    leeNum(num)
    mostrarTabla(num)

 fin_algoritmo
```

CODIFICACION	SALIDA DE PANTALLA

```cpp
#include <iostream>
using namespace std;
int leeNum();
void mostrarTabla(int);
int main(int argc, char *argv[]) {
    int num;
    num=leeNum();
    mostrarTabla(num);

    return 0;
}
int leeNum(){
    int num;
    do{
        cout<<"Ingrese número: ";
        cin>>num;
    }while(num<=0);
    return num;
}
void mostrarTabla(int num){
    int i;
    cout<<"Tabla de multiplicar del numero : "<<num<<endl;
    for(i=1;i<=12;i++)
        cout<<num<<" * "<<i <<" = "<<num*i<<endl;
}
```

```
Ingrese número: 5
Tabla de multiplicar del número: 5
5 * 1 = 5
5 * 2 = 10
5 * 3 = 15
5 * 4 = 20
5 * 5 = 25
5 * 6 = 30
5 * 7 = 35
5 * 8 = 40
5 * 9 = 45
5 * 10 = 50
5 * 11 = 55
5 * 12 = 60
```

Ejercicios resueltos

06. Desarrollar una aplicación que permita calcular el MCD de dos números enteros ingresados. Use Funciones.

PSEUDOCODIGO

```
entero mcd(entero x, entero y)
    entero d, p
    d ← 2
    p ← 1
    mientras d<=x y d<=y hacer
        si x mod d = 0 y y mod d =0 entonces
            p ← p * d
            x ← x / d
            y ← y / d
        sino
            d ← d +1
        fin_si
    fin_mientras
    retornar p
fin_mcd
```

PSEUDOCODIGO

```
Algoritmo Maximo_Comun_Divisor

    entero x, y
    hacer
        escribir "Ingrese 2 números: "
        leer x, y
    mientras x<=0 o y<=0
    escribir " El m.c.d es: ", mcd(x, y)

fin_algoritmo
```

CODIFICACION	SALIDA DE PANTALLA

```cpp
#include <iostream>
using namespace std;
int mcd(int x, int y);
int main(int argc, char *argv[]) {
    int x,y;
    do{
        cout<<"Ingrese 2 números: ";
        cin>>x>>y;
    }while(x<=0 || y<=0);
    cout<<"El M.C.D. es "<<mcd(x,y)<<endl;

    return 0;
}
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;
}
```

Ingrese 2 números: 10 5

El M.C.D. es 5

07. Desarrollar una aplicación que permita ingresar un número entero positivo y reportar si es primo o no.

PSEUDOCODIGO

```
logico esPrimo(entero n)

    entero i, cd
    cd ← 0
    para i ← 1 hasta n inc 1 hacer
        si  n mod i = 0 entonces
            cd ← cd +1
        fin_si
    fin_para
    si cd = 2 entonces
        retornar verdadero
    sino
        retornar falso
    fin_si
fin_esPrimo
```

PSEUDOCODIGO

```
Algoritmo verificarPrimo

    entero num
    Hacer
        escribir "Ingrese número: "
        leer num
    mientras num<=0
    si esPrimo(num) entonces
          escribir "El número es primo"
    sino
          escribir "El número no es primo"
    fin_si

fin_algoritmo
```

CODIFICACION	SALIDA DE PANTALLA

```cpp
#include <iostream>
using namespace std;

bool esPrimo(int );
int main(int argc, char *argv[])
{
    int num;
    do{
        cout<<"Ingrese número : ";
        cin>>num;
    }while(num<=0);
    if(esPrimo(num))
        cout<<"El número es primo"<<endl;
    else
        cout<<"El número no es primo"<<endl;
    return 0;
}

bool esPrimo(int n)
{
    int i,cd=0;
    for(i=1;i<=n;i++)
    {
        if(n%i==0)
            cd++;
    }
    if(cd==2) return true;
    else return false;
}
```

Ingrese número: 23

El número es primo

Variables globales

El ámbito de las variables globales se extiende desde el punto en el que se definen hasta el final del programa. En otras palabras, si definimos una variable al principio del programa, cualquier función que forme parte de éste podrá utilizarla simplemente haciendo uso de su nombre.

La utilización de variables globales proporciona un mecanismo de intercambio de información entre funciones sin necesidad de utilizar argumentos.

Por otra parte, las variables globales mantienen el valor que se les ha asignado dentro de su ámbito, incluso después de finalizar las funciones que modifican dicho valor.

Variables globales

Debemos tener en cuenta que el uso de variables globales para el intercambio de informaciones entre funciones puede resultar útil en algunas situaciones (como cuando se desea transferir más de un valor desde una función), **pero su utilización podría llevarnos a programas de difícil interpretación y complejos de depurar. (No es recomendable usar variables globales)**

```cpp
#include <iostream>
using namespace std;

void funcion1();
void funcion2();
int variable;

int main(int argc, char *argv[]) {

    variable = 9;
    cout<<"Valor de la variable : "<<variable<<endl;
    funcion1();
    funcion2();
    cout<<"Valor de la variable : "<<variable<<endl;
    return 0;
}

void funcion1()
{
    cout<<"En la funcion1, la variable es : "<<variable<<endl;
}
void funcion2()
{
    variable++;
    cout<<"En la otra funcion la variable es : "<<variable<<endl;
}
```

Variables estáticas

Otro tipo de almacenamiento son las variables estáticas identificadas por la palabra reservada static.

Las variables estáticas pueden ser tanto locales como globales.

Una variable estática local, al igual que una variable automática, está únicamente asociada a la función en la que se declara con la salvedad de que su existencia es permanente.

En otras palabras, su contenido no se borra al finalizar la función, sino que mantiene su valor hasta el final del programa

```cpp
#include <iostream>
using namespace std;

void imprimeValor();

int main(int argc, char *argv[]) {

    for(int i=1;i<=5;i++)
        imprimeValor();
    return 0;
}

void imprimeValor()
{
    static int contador = 0;
    contador++;
    cout<<"El valor de contador es : "<<contador<<endl;
}
```