UNIDAD CURRICULAR: ALGORITMICA Y PROGRAMACIÓN

UNIDAD VI. PROGRAMACIÓN MODULAR

CONTENIDO:

UNIDAD VI

PROGRAMACIÓN MODULAR

FUNCIONES Y PROCEDIMIENTOS

Una función o procedimiento es un miniprograma dentro de un programa. Las funciones contienen un bloque de instrucciones, bajo un solo nombre, que un programa puede utilizar una o más veces para ejecutar dichas instrucciones. Las funciones ahorran espacio, reduciendo repeticiones y facilitando la programación, proporcionando un medio de dividir un proyecto grande en módulos pequeños más manejables.

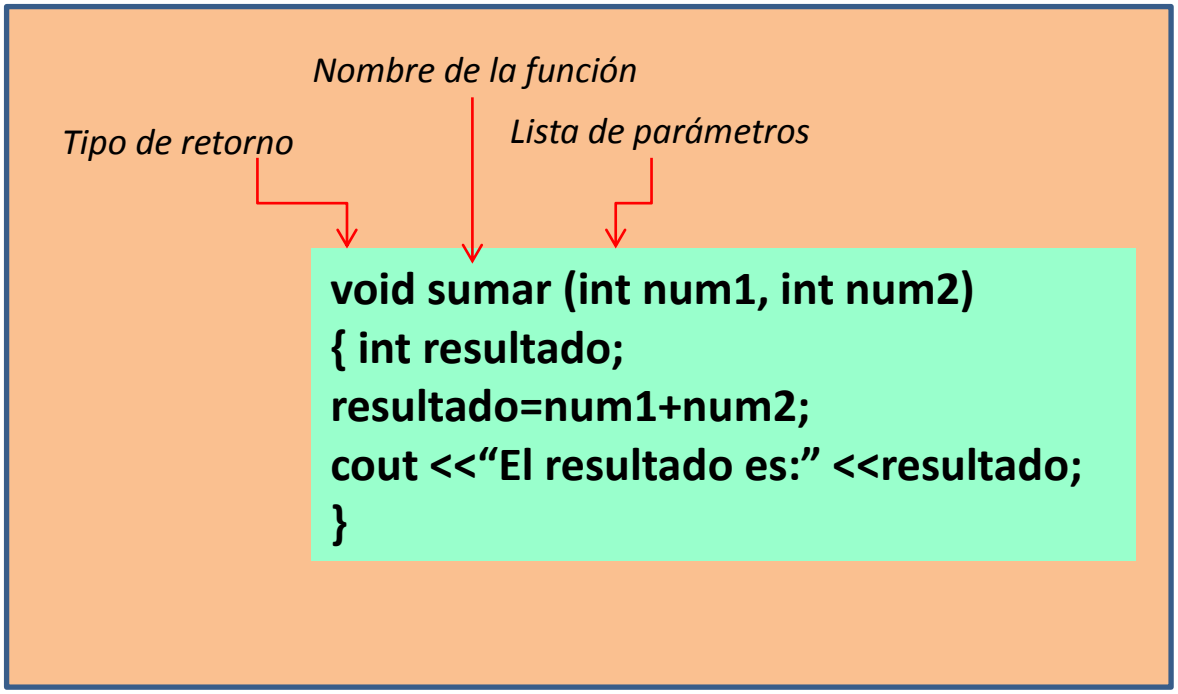
Un programa en C++ se compone de varias funciones que realizan una tarea específica.

Estructura de una función en C++:

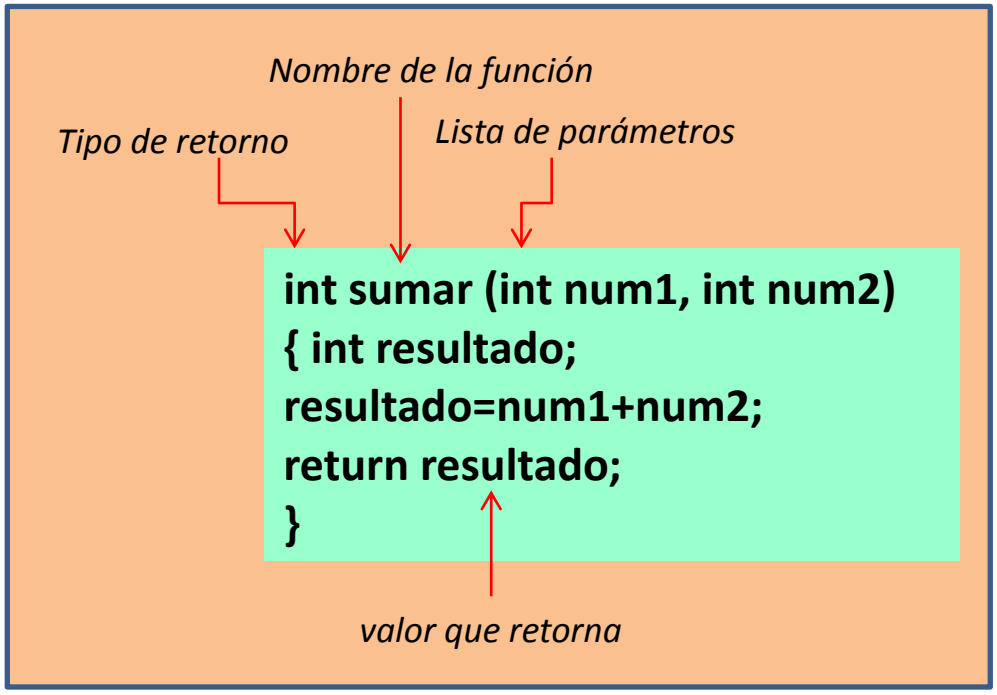
tipo_de_retorno nombre_de_la_función (*lista de parámetros*)
{ *bloque de instrucciones de la función*
return *valor_o_expresión*;
}

tipo_de_retorno: es el tipo de valor devuelto por la función o corresponde a la palabra reservada **void** si la función no retorna ningún valor.
nombre_de_la_función: identificador o nombre de la función
lista de parámetros: lista de declaraciones de los parámetros de la función separados por comas, con el formato siguiente: tipo_de_parámetro1 parámetro1, tipo_de parámetro2 parámetro2,……..
valor_o_expresión: valor que devuelve la función. El return se omite si la función no retorna un valor.

Ejemplo de una función que no retorna ningún valor:



Ahora modifiquemos la función para que retorne un valor:

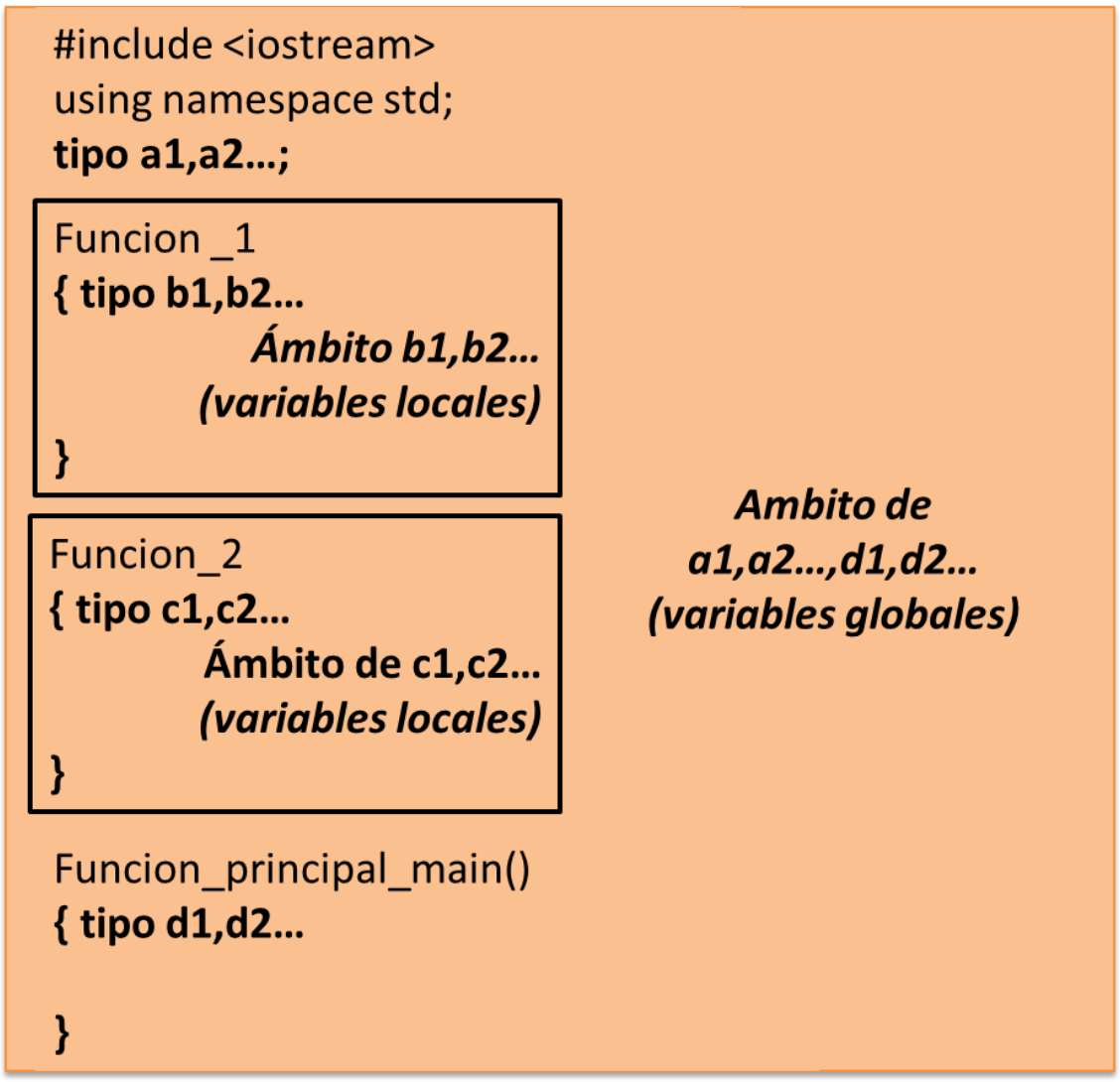


ÁMBITO DE VARIABLES: DATOS LOCALES Y GLOBALES.

El ámbito de una variable es la zona de un programa en el que es reconocida dicha variable. De acuerdo al ámbito las variables utilizadas en un programa se clasifican en: variables locales y variables globales.

Variable local: es aquella que está declarada y definida dentro de un subprograma o función y es distinta de cualquier otra variable, así tenga el mismo nombre, que esté declarada en cualquier parte del programa principal.

Variable global: es aquella que está declarada en el programa principal y es reconocida en cualquier parte del programa.



En el gráfico se observa que las variables a1,a2,…,d1,d2….., son variables globales cuyo ámbito comprende todo el cuerpo del programa, mientras que las variables b1,b2…,c1,c2,…., son variables locales que se reconocen sólo en la función donde fueron definidas.

LLAMADA DE UNA FUNCIÓN

Las funciones para ser ejecutadas deben ser llamadas o invocadas. Cualquier expresión puede contener una llamada a una función que redirigirá el control del programa a la función nombrada. La llamada a una función se puede realizar desde la función principal (main) o desde otra función.

Ejemplo:

```
#include <iostream>
using namespace std;
int n1,n2;
void sumar (int num1, int num2)
{ int  resultado;
resultado=num1+num2;
cout<<"El resultado es"<<resultado;
}
int main()
{ n1=7;
 n2=10;
 sumar(n1,n2)
return 0;
}
```

Dado que la función no retorna ningún valor, el llamado a la función se realiza solo colocando el nombre de la función con sus correspondientes parámetros. Ahora hagamos el llamado a una función que retorne un valor.

```
#include <iostream>
using namespace std;
int n1,n2,total;
int sumar (int num1, int num2)
{ int resultado;
resultado=num1+num2;
return resultado;
}
int main()
```

```
{ n1=7;
 n2=10;
 total=sumar(n1,n2)
cout<<"El resultado es"<<total;
return 0;
}
```

Para hacer el llamado a la función que retorna un valor, generalmente se le asigna a una variable la función con sus respectivos parámetros. Esta variable debe ser del mismo tipo que retorna la función.

Parámetros de una función:

C++ ofrece dos métodos para pasar variables (parámetros) entre funciones. Una función puede utilizar parámetros por valor y parámetros por referencia, o puede no tener parámetros.

Paso de parámetros por valor: significa que C++ compila la función y el código que llama a la función, la función recibe una copia de los valores de los parámetros. Si se cambia el valor de un parámetro variable local, el cambio sólo afecta a la función y no tiene efecto fuera de la función.

En el ejemplo anterior, se utilizan parámetros por valor.

Paso de parámetros por referencia: Se utiliza cuando una función modifica el valor del parámetro pasado y devuelve este parámetro modificado a la función que la invoca. En este caso, el compilador pasa la dirección de memoria del valor del parámetro a la función a través del operador **&.** Si este valor se modifica, queda almacenado en la misma dirección de la memoria, por lo que al retornar a la función que lo invoca dicha dirección de memoria contendrá el valor modificado.

C++ utiliza punteros para implementar parámetros por referencia. El uso de punteros se profundizará mas adelante. Por ahora se muestra el siguiente ejemplo de una función con parámetros por referencia:

Ejemplo:

```
#include <iostream>
using namespace std;
int num1,num2;
void cambiar (int *n1, int *n2)
```

```
{
*n1=8;
*n2=4;
}
int main()
{ num1=7;
 num2=10;
 cambiar(&num1,&num2)
cout<<"El valor de num1 es"<<num1;
cout<<"El valor de num2 es"<<num2;
return 0;
}
```

Es importante resaltar el llamado a la función:

cambiar(&num1,&num2)

Al realizar el llamado a la función se pasa la dirección de memoria de las variables num1 y num2 anteponiendole el operador **&**.

Por otra parte, en la implementación de la función

void cambiar (int *n1, int *n2)

Se antepone a cada parámetro el operador ***** para acceder a las variables referenciadas por las direcciones de las variables num1 y num2.

Al ejecutar el ejemplo anterior se mostrarían los siguientes mensajes:

El valor de num1 es 8

El valor de num2 es 4

Es evidente que las variables num1 y num2 cambiaron su valor dentro de la función ***cambiar*** y mantienen dicho cambio fuera de la función.

EJERCICIOS RESUELTOS

a) Realice una función que retorne el área de un cuadrado y reciba como parámetro el valor del lado.

```
float area_cuadrado (float lado)
{ float area;
area=lado*lado;
```

```
return area;}
```

b) Realice una función que tenga como parámetros dos números e indique si son iguales o cual es mayor.

```
void comparar_numeros (int n1, int n2)
{ if (n1>n2)
    cout<< "El número " << n1 << " es mayor que " << n2;
 else
     if (n2>n1)
        cout<< "El número " << n2 << " es mayor que " << n1;
    else
        cout<< "El número " << n1 << " es igual a " << n2;
}
```

c) Realice una función que reciba como parámetro el sueldo actual de un empleado y lo actualice incrementándole un 20%.

```
void aumento_sueldo (float *sueldo)
{ float aumento;
  aumento=*sueldo*20/100;
  *sueldo=*sueldo+aumento;
}
```

d) Realice una función que retorne el salario de un trabajador dado un número de horas trabajadas y el salario por hora. Las horas que superan las 40 horas semanales se pagarán como extras con un salario hora del 50% adicional al salario hora normal.

```
float calcular_salario (float horas_trabajadas, float salario_por_hora)
{ float salario,horas_extras;
  if (horas_trabajadas <=40)
    { salario=horas_trabajadas * salario_por_hora;}
  else
    { horas_extras=horas_trabajadas-40;
      salario= 40*salario_por_hora + horas_extras * salario_por_hora +
```

```
                horas_extras * salario_por_hora*50/100; }
 return salario;}
```

e) Realice una función que retorne el área y el perímetro de un rectángulo. (Area= base*altura y Perimetro= base * 2 + altura*2)

```
void area_perimetro_rect (float base, float altura, float *area, float *perimetro)
{*area= base * altura;
  *perimetro= base * 2 + altura * 2;}
```

REFERENCIAS BIBLIOGRÁFICAS

Bassard, G y Bratley, P. (2010). Fundamentos de algoritmia. Prentice-Hall.

Joyanes, L. (2008). Fundamentos de programación. Algoritmos , Estructuras de datos y objetos. Mc Graw Hill. Tercera edición.

Joyanes, L. y Zahonero, I. (2005). Programación en C. Metodología, algoritmos y Estructura de datos. Mc Graw Hill. Segunda Edición

.