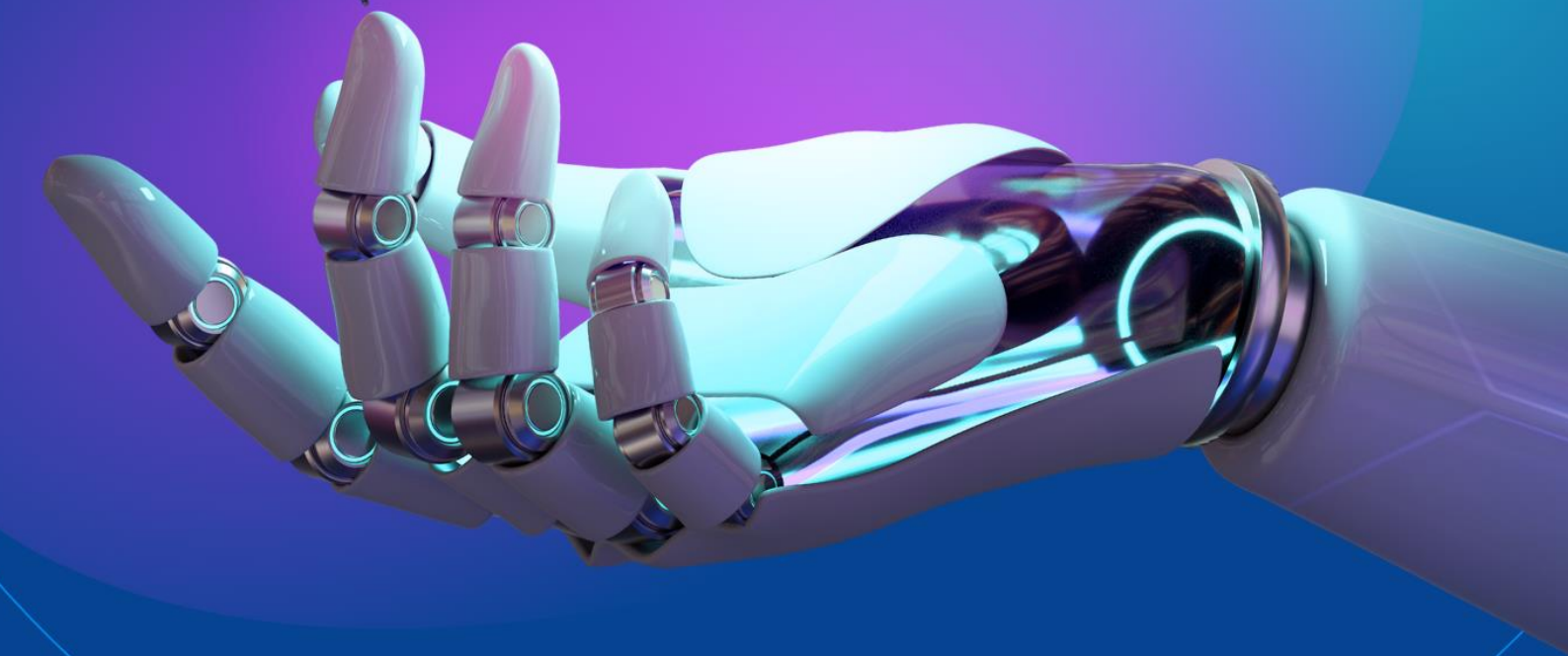
MODULARIDAD EN C++

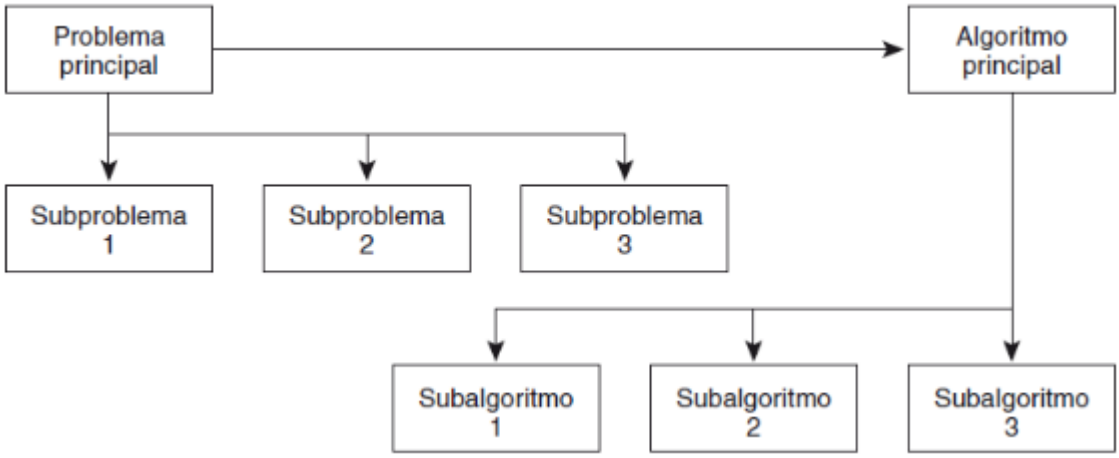
- FUNCIONES
- PROCEDIMIENTOS
- RECURSIVIDAD



Estructura de Datos

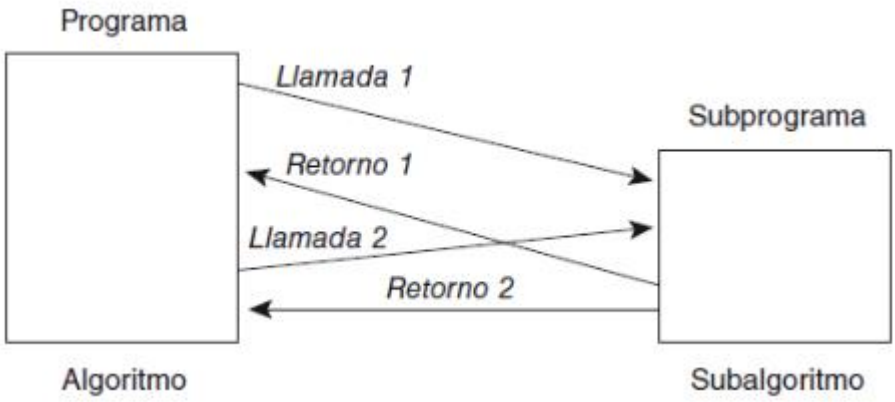
SUBPROGRAMAS

Un método muy útil para solucionar un problema complejo es dividirlo en subproblemas, problemas más sencillos y a continuación dividir estos subproblemas en otros más simples, hasta que los problemas más pequeños sean fáciles de resolver. Esta técnica de dividir el problema principal en subproblemas se suele denominar **"divide y vencerás".** Este método de diseñar la solución de un problema principal obteniendo las soluciones de sus subproblemas se conoce como diseño descendente (top-down). Se denomina descendente, ya que se inicia en la parte superior con un problema general y se termina con varios subproblemas de ese problema general y las soluciones a esos subproblemas. Luego, las partes en que se divide un programa deben poder desarrollarse independientemente entre sí. Las soluciones de un diseño descendente pueden implementarse fácilmente en lenguajes de programación y se los denomina subprogramas o sub-algoritmos si se emplean desde el concepto algorítmico.

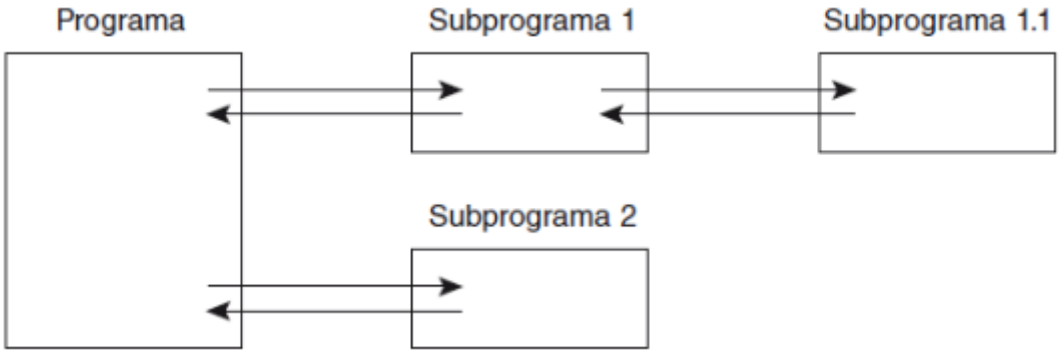


El problema principal se soluciona por el correspondiente programa o algoritmo principal, mientras que la solución de los subproblemas será a través de subprogramas, conocidos como procedimientos o funciones. Un subprograma es un como un mini algoritmo, que recibe los datos, necesarios para realizar una tarea, desde el programa y devuelve los resultados de esa tarea.

Cada vez que el subprograma es invocado, el algoritmo va al subprograma invocado y se realiza el subprograma, y a su vez, un subprograma puede llamar a otros subprogramas.



Un programa con un subprograma: función y procedimiento



Un programa con diferentes niveles de subprogramas

Cabe mencionar que a los módulos también se les conoce como, sub algoritmos, subprogramas y sub rutinas. A nivel general podemos clasificar en dos:

- Funciones
- Procedimientos

Diferencias entre funciones y procedimientos:

Estos subprogramas son similares, aunque presentan diferencias entre ellos.

1. Las funciones devuelven un sólo valor al algoritmo principal. Los procedimientos pueden devolver cero, uno o más valores.
2. A un nombre de procedimiento no se puede asignar un valor y por consiguiente ningún tipo está asociado con un nombre de procedimiento.
3. Una función se referencia utilizando su nombre en una expresión mientras que un procedimiento se referencia por una llamada o invocación al mismo.

FUNCIONES

Una función es un módulo de código con un nombre asociado que realiza una serie de tareas y **devuelve un valor**. Desde el punto de vista matemático, una función es una operación que toma uno o a varios operandos, y devuelve un resultado. Y desde el punto de vista algorítmico, es un subalgoritmo que toma uno o varios parámetros como entrada y devuelve a la salida un único resultado. f(p1,p2,...,pn) donde p1, p2, ..., pn son los parámetros y f es el nombre de la función. Cabe recordar que para el uso de las funciones en muchas ocasiones es necesario usar variables globales.

Declaración de una función:

Para declarar una función en C++, necesitas definir el **tipo de retorno de la función**, el **nombre de la función** y los **parámetros que la función aceptará**. A continuación, te proporciono una estructura básica de una declaración de función:

Sintaxis:

Algoritmo	C++

//Función
Sub nombre_módulo(parametro)
//procesos
retorna valor
Fin sub
	//Función
	`int suma(int a, int b) {`
	`return a + b;`
	`}`

Invocación a las Funciones:

Una función puede ser llamada de la siguiente forma:

nombre_función (Argumentos)

- nombre_función: función que va a llamar.
- argumentos: constantes, variables, expresiones.

Cada vez que se llama a una función desde el algoritmo principal se establece automáticamente una correspondencia entre los argumentos y los parámetros. Debe haber exactamente el mismo número de parámetros que de argumentos en la declaración de la función y se presupone una correspondencia uno a uno de izquierda a derecha entre los argumentos y los parámetros. Además, cuando se llama a la función está va a devolver el resultado de las acciones realizadas en la función (variable de retorno) este resultado debe ser “atrapado” en el algoritmo. Ya sea para usarlo o solo para mostrarlo, por lo que al llamar una función debemos, o asignarle el resultado a una variable o concatenar el llamado de una función con un escribir “<<cout”

Ejemplo:

variable = nombre_función(argumentos).

Escribir nombre_función(argumentos)

Una llamada a la función implica los siguientes pasos:

1. A cada argumento se le asigna el valor real de su correspondiente parámetro.
2. Se ejecuta el cuerpo de acciones de la función.
3. Se devuelve el valor de la función y se retorna al punto de llamada.

Entonces para resumir se puede decir que la función tiene cinco componentes importantes:

- el **identificador**: va a ser el nombre de la función, mediante el cual la invocaremos.
- los **parámetros** son los valores que recibe la función para realizar una tarea.
- los **argumentos**, son los valores que envía el algoritmo a la función.
- las **acciones de la función**, son las operaciones que hace la función.
- **valor de retorno** (o el resultado), es el valor final que entrega la función. La función devuelve un único valor.

Ámbito: Variables Locales y Globales

Las variables utilizadas en los programas principales y subprogramas se clasifican en dos tipos: variables locales y variables globales.

Una **variable local** es aquella que está declarada y definida dentro de un subprograma, en el sentido de que está dentro de ese subprograma, es distinta de las variables con el mismo nombre declaradas en cualquier parte del programa principal y son variables a las que el algoritmo principal no puede acceder de manera directa. El significado de una variable se confina al procedimiento en el que está declarada. Cuando otro subprograma utiliza el mismo nombre se refiere a una posición diferente en memoria. Se dice que tales variables son locales al subprograma en el que están declaradas.

Una **variable global** es aquella que está declarada en el programa principal, del que dependen todos los subprogramas y a las que pueden acceder los subprogramas, a través del paso de argumento. La parte del programa en que una variable se define se conoce como ámbito o alcance (scope, en inglés). El uso de variables locales tiene muchas ventajas. En particular, hace a los subprogramas independientes, siendo solo la comunicación entre el programa principal y los subprogramas a través de la lista de parámetros. Una variable local a un subprograma no tiene ningún significado en otros subprogramas. Si un subprograma asigna un valor a una de sus variables locales, este valor no es accesible a otros programas, es decir, no pueden utilizar este valor. A veces, también es necesario que una variable tenga el mismo nombre en diferentes subprogramas. Por el contrario, las variables globales tienen la ventaja de compartir información de diferentes subprogramas sin la necesidad de ser pasados como

argumento.

PROCEDIMIENTOS

Aunque las funciones son herramientas de programación muy útiles para la resolución de problemas, con frecuencia se requieren subprogramas que no retornen ninguna información o se encarguen de imprimir información. En estas situaciones la función no es apropiada y se necesita disponer del otro tipo de subprograma: el procedimiento. Un procedimiento es un subprograma o un sub algoritmo que ejecuta una determinada tarea, pero que tras ejecutar no necesariamente debe devolver un valor aun cuando tengan argumentos. En realidad, los argumentos son parámetros (variables) de entrada cuando ejecutas un programa.

Declaración de un procedimiento:

En C++, un procedimiento se define como una función que no devuelve ningún valor, es decir, su tipo de retorno es `void`. A continuación, te presento la estructura básica de la declaración y definición de un procedimiento en C++:

Sintaxis:

Algoritmo	C++

//procedimiento
SubProceso nombre(parámetros)
<acciones>
FinSubProceso
	//procedimiento
	`void saludar() {`
	`cout << "Hola, mundo!" <<endl;`
	`}`

Los parámetros tienen el mismo significado que en las funciones.

Invocación a un procedimiento:

Un procedimiento puede ser llamado de la siguiente forma:

nombre_procedimiento (Argumentos)

- nombre_procedimiento: procedimiento que se va a llamar.
- argumentos: constantes, variables, expresiones.

Comunicación con Subprogramas: Paso de Argumentos:

Cuando un programa llama a un subprograma, la información se comunica a través de la lista de parámetros y se establece una correspondencia automática entre los parámetros y los argumentos. Los parámetros son “sustituidos” o “utilizados” en lugar de los argumentos.

La declaración del subprograma se hace con:

Subproceso nombre (PA1, PA2, ..., PAn)

<acciones>

FinSubProceso

y la llamada al subprograma con:

nombre (AR1, ARG2,..., ARGn)

donde **PA1, PA2, ..., PAn** son los parámetros y **ARG1, ARG2, ..., ARGn** son los argumentos.

Cuando nosotros decidimos los parámetros que va a necesitar nuestro subprograma, también podemos decidir cuál va a ser el comportamiento de los argumentos en nuestro subprograma cuando lo invoquemos y se los pasemos por paréntesis. Esto va a afectar directamente a los argumentos y no al resultado final del subprograma. Para esto existen dos tipos más empleados para realizar el paso de argumentos, el paso por valor y el paso por referencia.

Paso por Valor

Los argumentos se tratan como variables locales y los valores de dichos argumentos se proporcionan copiando los valores de los argumentos originales. Los parámetros (locales a la función o procedimiento) reciben como valores iniciales una copia de los valores de los argumentos y con ello se ejecutan las acciones descritas en el subprograma. Aunque el paso por valor es sencillo, tiene una limitación acusada: no existe ninguna otra conexión con los parámetros, y entonces los cambios que se produzcan dentro del subprograma no producen cambios en los argumentos originales y, por consiguiente, no se pueden poner argumentos como valores de retorno. El argumento actual no puede modificarse por el subprograma.

Paso por Referencia

En numerosas ocasiones se requiere que ciertos argumentos sirvan como argumentos de salida, es decir, se devuelvan los resultados al programa que llama. Este método se denomina paso por referencia o también de llamada por dirección o variable. El programa que llama pasa al subprograma la dirección del argumento actual (que está en el programa que llama). Una referencia al correspondiente argumento se trata como una referencia a la posición de memoria, cuya dirección se ha pasado. Entonces una variable pasada como argumento real es compartida, es decir, se puede modificar directamente por el subprograma. La característica de este método se debe a su simplicidad y su analogía directa con la idea de que las variables tienen una posición de memoria asignada desde la cual se pueden obtener o actualizar sus valores. El área de almacenamiento (direcciones de memoria) se utiliza para pasar información de entrada y/o salida; en ambas direcciones. En este método los argumentos son de entrada/salida y los argumentos se denominan argumentos variables.

FUNCIONES RECURSIVAS

La recursión es una técnica de programación en la que una función se llama a sí misma para resolver un problema. Un problema se divide en subproblemas más pequeños del mismo tipo, y la función resuelve esos subproblemas de manera recursiva.

Componentes de una Función Recursiva:

- **Caso base:** La condición que detiene las llamadas recursivas.
- **Caso recursivo:** La parte de la función donde se hace la llamada recursiva.