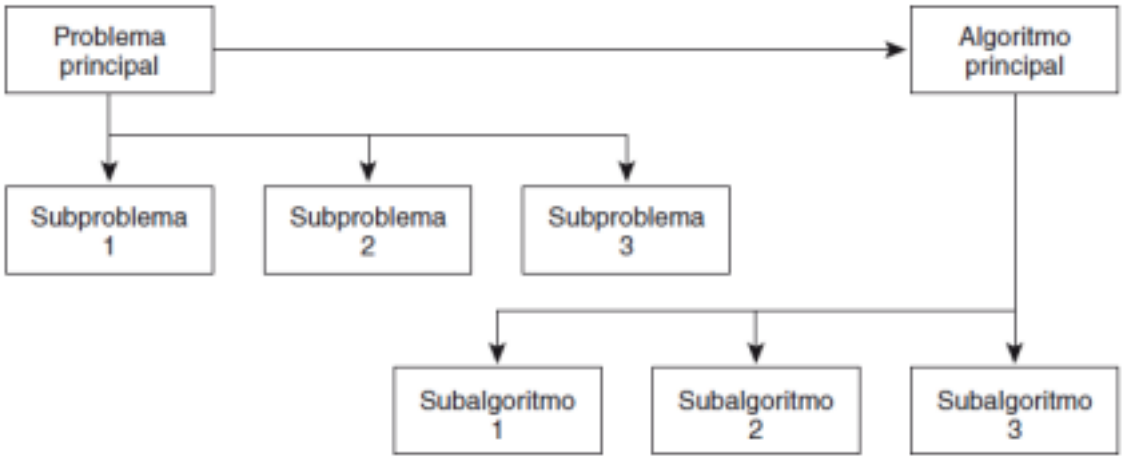
Material descargable

Fundamentos de la Programación

¿Qué es el diseño descendente (top-down)?

Hasta el momento, has trabajado tus operaciones dentro del algoritmo principal, en un único bloque de código. Este tipo de diseño puede provocar repetición en el código, además de dificultar el mantenimiento del mismo.

Para simplificar los inconvenientes surgidos por este tipo de diseño, existe el **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 los soluciones a esos subproblemas. Luego, las partes en que se divide un programa deben poder desarrollarse independientemente entre sí.



¿Cómo implementar los subproblemas en PSeInt?

Un subproblema o subprograma se refiere a una porción de código dentro de un programa más grande que realiza una tarea específica. En PSeINT, **existen dos formas de implementar esta técnica**: **las funciones y los procedimientos**. Ambos pueden ser invocados desde cualquier parte del programa principal para ejecutar la tarea que han sido diseñados para realizar. Esto fomenta la reutilización del código y contribuye a organizar el programa en unidades más pequeñas y manejables.

Tanto las funciones como los procedimientos pueden recibir datos como entrada, procesarlos de acuerdo con una serie de instrucciones internas y, opcionalmente,

devolver un resultado como salida. Esta capacidad los hace muy útiles para dividir problemas complejos en problemas más pequeños y manejables, facilitando así el desarrollo y la depuración del software.

En PSeInt, **la diferencia entre un procedimiento y una función radica principalmente en si devuelven o no un valor como resultado**. Si devuelve un valor, se les suele llamar funciones; si no devuelven ningún valor, se les suele llamar procedimientos o subprogramas. Sin embargo, ambos cumplen la misma función de encapsular una serie de instrucciones para llevar a cabo una tarea específica.

¿Qué son las funciones?

Las funciones son un tipo de subprograma. Son bloques de código que se pueden invocar múltiples veces. Deben tener un propósito específico para resolver un problema particular dentro del programa general. **Una función debe devolver siempre un resultado**, el cual puede ser de cualquier tipo de dato, pero siempre debe retornar un valor que será almacenado en la variable de retorno.

¿CÓMO SE DECLARA UNA FUNCIÓN?

Una función se define fuera del algoritmo principal y sigue una serie de pasos específicos para su creación. Funciona como un subalgoritmo o subprograma, con una estructura similar a la de los algoritmos. Comienza con la palabra reservada "Función" y termina con "FinFunción", al igual que un algoritmo. La definición de la función incluye una cabecera que especifica el dato que la función devolverá. Este valor devuelto se define dentro del cuerpo de la función y debe representar el resultado de una tarea específica relacionada con el problema en cuestión.

Luego de la palabra "Función", se coloca el nombre de la función, seguido de paréntesis que contienen los parámetros de la función. Los parámetros son los datos que la función necesita recibir del algoritmo para llevar a cabo su tarea. Estos parámetros se especifican escribiendo el nombre de la variable a recibir, sin incluir su tipo de dato. Si se necesitan más de un parámetro, se separan por comas. Aunque no son obligatorios, es poco común tener una función sin parámetros.

Finalmente, el cuerpo de la función consiste en una serie de instrucciones que se ejecutan para asignar un valor al nombre de la función. Estas acciones determinan el valor específico que la función devolverá al programa principal.

Cuando se invoca o llama a la función desde el algoritmo principal, se utiliza su nombre seguido de una lista de argumentos que deben coincidir en cantidad, tipo y orden con

los parámetros definidos en la función. Los argumentos son las variables o valores que se pasan a la función durante la llamada.

En resumen, formato general de una función:

- **Palabra reservada:** Funcion
- **Variable retorno**: Donde la función almacena el resultado.
- **Nombre de la función**: Nombre que utilizaremos para invocar la función
- **Conjunto de Parámetros** (opcional): Puede tener una cantidad variable de parámetros, que permiten que el programa se comunique, enviando información a través de ellos.

```
Funcion variable_de_retorno <- Nombre ( Argumentos )

Fin Funcion
```

Dentro de la función:

- **Variable retorno**: Debe ser definida, es decir, se debe indicar qué tipo de dato retorna esa función.
- Se pueden implementar diversas acciones dentro del bloque de código, pero solo se retorna el último valor almacenado en la variable creada para dicho fin. ¡Recordar asignar este valor

Ejemplo Uso de Funciones en PSeInt

```
Algoritmo EjercicioEjemplo
    Definir varNumero1,varNumero2, resultado Como Entero

    Escribir "Ingrese dos numeros para realizar la multiplicación"
    Leer varNumero1,varNumero2

    resultado = calcularMultiplo(varNumero1,varNumero2)
    Escribir " La multiplicación de ambos numeros es: " , resultado

FinAlgoritmo

Funcion varMulti = calcularMultiplo(varNumero1 Por Valor,varNumero2 Por Valor)
    Definir varMulti como Entero
    varMulti = varNumero1 * varNumero2
FinFuncion
```

Desglose del ejemplo:

- **Descripción del Algoritmo** :El algoritmo comienza definiendo tres variables: **varNumero1**, **varNumero2** y **resultado**, todas ellas de tipo entero. Luego, solicita al usuario que ingrese dos números mediante el mensaje "Ingrese dos números para realizar la multiplicación". Después de que el usuario ingresa los números, **el algoritmo llama a la función calcularMultiplo pasando varNumero1 y varNumero2 como argumentos. El resultado de esta función se almacena en la variable resultado.** Finalmente, imprime el resultado de la multiplicación.
- **Función calcularMultiplo**: La función **calcularMultiplo** recibe dos parámetros por valor, varNumero1 y varNumero2, que representan los dos números que se multiplicarán. Dentro de la función, se declara una variable local varMulti como entero para almacenar el resultado de la multiplicación. Luego, se realiza la multiplicación de varNumero1 y varNumero2, y se asigna el resultado a varMulti.

Beneficios de Usar Funciones

- **Modularidad**: Divide el código en partes más manejables y entendibles.
- **Reutilización**: Las funciones pueden ser reutilizadas en diferentes partes del programa o en otros programas, la cantidad de veces que se necesite.
- **Mantenimiento**: Facilita el mantenimiento y la actualización del código.
- **Claridad**: Mejora la claridad y la legibilidad del código, haciendo que el propósito de cada parte sea más evidente.

Comunicación con Subprogramas: Paso de Argumentos

Cuando un programa llama a un subprograma, la comunicación se realiza a través de la lista de parámetros, estableciendo una correspondencia automática entre estos y los argumentos. Los parámetros actúan como marcadores de posición que serán "sustituidos" o "utilizados" por los argumentos proporcionados.

La declaración de un subprograma se realiza mediante la sintaxis:

```
Subproceso nombre (PA1, PA2, ..., PAn)
    <acciones>
FinSubproceso
```

Mientras que la llamada al subprograma se realiza así:

```
nombre (ARG1, ARG2,..., ARGn)
```

Donde PA1, PA2, ..., PAn representan los parámetros y ARG1, ARG2, ..., ARGn son los argumentos.

Para comprender esto en un contexto más cotidiano, podemos imaginar los subprogramas como formularios en blanco que requieren ser completados. Si un formulario solicita los campos 'nombre' y 'apellido', estos serán los parámetros del subprograma. Al completar el formulario con nuestros datos, proporcionamos nuestro nombre y apellido como argumentos. Esto permite que múltiples personas completen el mismo formulario, cada una con su propia información.

Al diseñar nuestros subprogramas, también podemos determinar cómo se comportarán los argumentos cuando los pasemos por paréntesis al invocar el subprograma. Esto afecta directamente a los argumentos, no al resultado final del subprograma.

Existen dos tipos comunes de paso de argumentos: paso por valor y paso por referencia.

1. **Paso por Valor :**En este método, los argumentos se tratan como variables locales y se proporciona una copia de sus valores originales a los parámetros locales del subprograma. Los cambios realizados dentro del subprograma no afectan a los argumentos originales, ya que estos últimos no están vinculados de ninguna otra manera a los parámetros. En PSeInt, todas las variables pasadas como argumentos se consideran "Por Valor" por defecto, a menos que se especifique explícitamente lo contrario.
2. **Paso por Referencia:** Este método implica que ciertos argumentos actúen como variables de salida, es decir, los resultados se devuelven al programa que llama. El programa que llama proporciona al subprograma la dirección del argumento actual, permitiendo que este último sea modificado directamente por el subprograma. En el paso por referencia, los argumentos son de entrada/salida, y se denominan argumentos variables debido a que pueden ser modificados por el subprograma. Este método se basa en la idea de que las variables tienen una posición de memoria asignada desde la cual pueden obtener o actualizar sus valores. Las direcciones de memoria se utilizan para pasar información tanto de entrada como de salida entre el programa principal y el subprograma.

RECUERDA: Los **parámetros** son variables en la definición de una función, mientras que los **argumentos** son los valores reales que se pasan a la función cuando se la invoca. La cantidad, tipo y orden de los argumentos enviados al invocar la función deben ser coherentes con los declarados en la misma.