CURSO DE PROGRAMACIÓN FULL STACK

SUBPROGRAMAS CON PSEINT

EGG

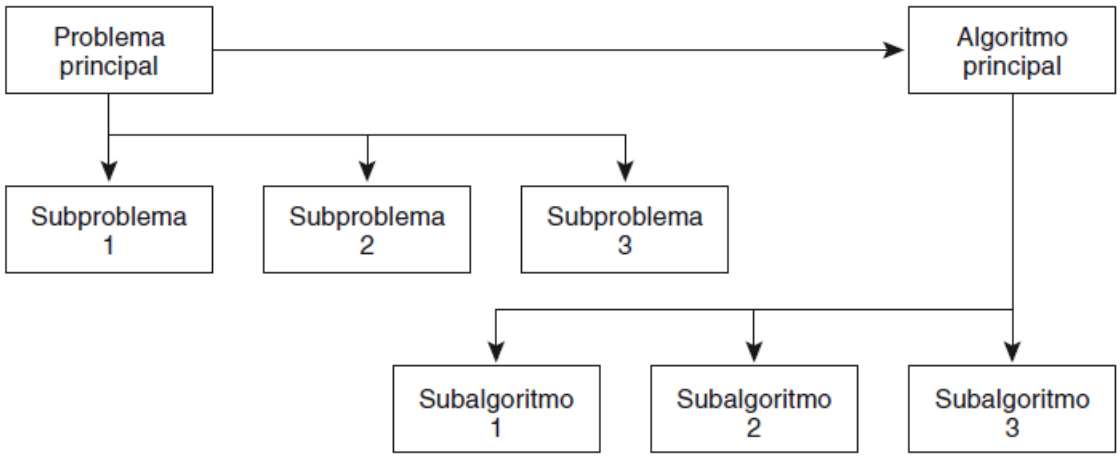
GUÍA DE SUBPROGRAMAS

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".

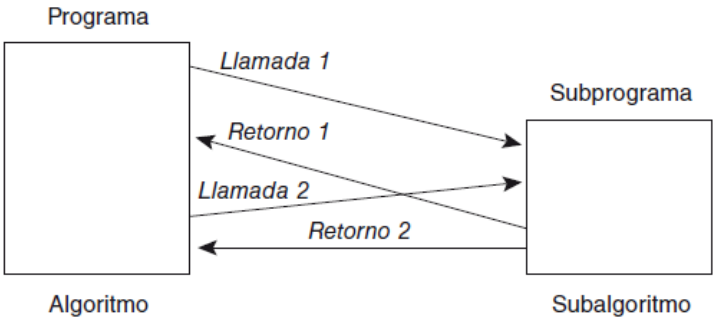
Este método de diseñar la solución de un problema principal obteniendo las soluciones de sus subproblemas se conoce como diseño descendente (top-down). Se denomina descendente, ya que se inicia en la parte superior con un problema general y se termina con varios subproblemas de ese problema general y los 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.

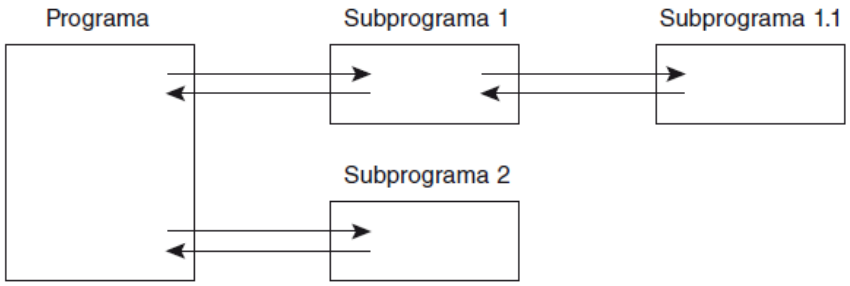


El problema principal se soluciona por el correspondiente programa o algoritmo principal, mientras que la solución de los subproblemas será a través de subprogramas, conocidos como **procedimientos** o **funciones**. Un subprograma es un como un mini algoritmo, que recibe los *datos,* necesarios para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

FUNCIONES

Matemáticamente una función es una operación que toma uno o más valores llamados *argumentos* y produce un resultado.

Cada función se evoca utilizando su nombre en una expresión con los argumentos encerrados entre paréntesis. A una función no se le llama explícitamente, sino que se le invoca o referencia mediante un nombre y una lista de parámetros.

Declaración de una función

Cada función se crea fuera de nuestro algoritmo y requiere de una serie de pasos que la definen. Una función como tal subalgoritmo o subprograma tiene una constitución similar a los algoritmos. Esta comenzará con la palabra reservada Función y termina con la palabra FinFunción al igual que un Algoritmo. Al lado de nuestra palabra Funcion, comenzaremos la creación de nuestra función, esta constará de una cabecera que comenzará con un nombre para el valor devuelto por la función. El valor devuelto será una variable que definiremos dentro del cuerpo de nuestra función, ahí la daremos un tipo de dato. Este valor devuelto debe ser el resultado de la tarea que hemos dividido del problema general.

Después, va a ir el nombre de nuestra función y a continuación, entre paréntesis los parámetros de dicha función. Los parámetros van a ser los datos que necesitamos que nos envíe el algoritmo para realizar el subproblema en cuestión. Estos se escriben poniendo el nombre de la variable a recibir, sin su tipo de dato, y si quisiéramos pasar más de una variable, los separamos con comas. Los parámetros no son obligatorios a la hora de usar un subprograma, podemos tener una función sin parámetros, aunque es poco común.

Por ultimo, irá el cuerpo de la función, que será una serie de acciones o instrucciones cuya ejecución hará que se asigne un valor al nombre de la función. Esto determina el valor particular del resultado que ha de devolverse al programa llamador.

El algoritmo o programa llama o invoca a la función con el nombre de esta última en una expresión seguida de una lista de argumentos que deben coincidir en cantidad, tipo y orden con los parámetros de la función. Se denominan argumentos a las variables o valores declarados en el algoritmo. Cuando se realiza una llamada a la función, los "valores" pasados o enviados a la función se denominan argumentos.

SINTAXIS

```
Funcion variable_de_retorno <- Nombre (Parámetros)

      Definir variable_de_retorno como Tipo de Dato

      <acciones>   //cuerpo de la función

FinFuncion
```

- Parámetros: uno o más parámetros de la siguiente forma: (parámetro 1 [Por Valor/Por Referencia], parámetro 2 [Por Valor/Por Referencia],...]).
- Nombre asociado con la función: que será un nombre de identificador válido.
- <acciones>: instrucciones que constituyen la definición de la función y que debe contener alguna instrucción mediante la cual se asigne un valor a la variable_de_retorno.
- Variable_de_retorno: contiene el resultado que devuelve la función que debe coincidir con el tipo de dato de retorno.

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

Ejemplo:

variable = nombre_funcion(argumentos).

Escribir nombre_funcion(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*.

Ejemplo para abrir en Pseint: Funcion.

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 que ejecuta un proceso específico. Ningún valor está asociado con el nombre del procedimiento; por consiguiente, no puede ocurrir en una expresión. Un procedimiento se llama escribiendo su nombre. Cuando se invoca el procedimiento, los pasos que lo definen se ejecutan y a continuación se devuelve el control al programa que le llamó.

SINTAXIS

```
SubProceso Nombre (parámetros)

      <acciones>

FinSubProceso
```

Los parámetros tienen el mismo significado que en las funciones.

Invocación a un Subprograma

Un procedimiento puede ser llamado de la siguiente forma:

```
nombre(argumentos)
```

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

Ejemplo para abrir en Pseint: Procedimiento.

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 o algoritmo 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/algoritmo 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

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 cual 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.
En PSeInt todas las variables que pasemos como argumentos pasan por defecto "Por Valor" sino se especifica lo contrario explícitamente.

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.

Recursión

Una función o procedimiento que se puede llamar a sí mismo se llama recursivo. La recursión (recursividad) es una herramienta muy potente en algunas aplicaciones, sobre todo de cálculo. La recursión puede ser utilizada como una alternativa a la repetición o estructura repetitiva. El uso de la recursión es particularmente idóneo para la solución de aquellos problemas que pueden definirse de modo natural en términos recursivos.
La escritura de un procedimiento o función recursiva es similar a sus homónimos no recursivos; sin embargo, para evitar que la recursión continúe indefinidamente es preciso incluir una condición de terminación, que suele hacerse con una estructura condicional.

PREGUNTAS DE APRENDIZAJE

1. **Un subprograma es:**
   a) Un código especial que se utiliza para resolver distintos tipos de problemas.
   b) Un método de solucionar un problema complejo dividiéndolo en subproblemas.
   c) Un método que siempre debe retornar algún resultado.
   d) Ninguna de las anteriores.

2. **Un argumento es:**
   a) El valor enviado por el programa principal al subprograma.
   b) Una variable local.
   c) El valor que recibe el subprograma enviado del programa principal.
   d) Ninguna de las anteriores.

3. **Un parámetro es:**
   a) El valor enviado por el programa principal al subprograma.
   b) Una variable global.
   c) El valor que recibe el subprograma enviado del programa principal.
   d) Ninguna de las anteriores.

4. **Una variable puede pasarse como argumento a un subprograma:**
   a) Sólo por valor
   b) Sólo por referencia
   c) Por valor y por referencia
   d) No puede pasarse como argumento

5. **¿Qué características tienen los elementos locales?**
   a) Son visibles en su ámbito y fuera
   b) Son invisibles en su ámbito y fuera
   c) Son invisibles en su ámbito y visibles fuera
   d) Son visibles en su ámbito e invisibles fuera

6. **¿Qué características tienen los elementos globales?**
   a) Son visibles en su ámbito y fuera
   b) Son invisibles en su ámbito y fuera
   c) Son visibles en su ámbito e invisibles fuera
   d) Son invisibles en su ámbito y visibles fuera

7. **Una función de un programa siempre debe**
   a) Recibir al menos un argumento
   b) Ser llamado en el algoritmo principal
   c) Ser recursivo
   d) Ninguna de las anteriores

8. **Cuando se escribe una función PSeInt**
   a) Es necesario definir el tipo de dato de la variable de retorno
   b) No es necesario definir el tipo de dato de la variable de retorno
   c) Es indiferente si el tipo de dato de la variable de retorno se define o no
   d) Ninguna de las anteriores