UT3 Programas y subprogramas

RESULTADO DE APRENDIZAJE
1. Reconoce la estructura de un programa informático, identificando y relacionando los elementos propios del lenguaje de programación utilizado

Tabla de contenido

UT3 Programas y subprogramas 1
1. PROGRAMACIÓN MODULAR 2
Lenguajes de programación estructurados y modulares 3
2. FUNCIONES Y PROCEDIMIENTOS 5
2.1. Introducción 5
2.2. Programas y subprogramas 5
2.3. Parámetros 6
Paso de parámetros 6
2.4. Funciones 6
2.5. Procedimientos 7
3. ACOPLAMIENTO Y COHESIÓN 8
4. RECURSIVIDAD 8
5. Bibliografía 9

1. PROGRAMACIÓN MODULAR

Un paradigma de programación es un enfoque particular para la construcción del software. Define el conjunto de reglas, patrones y estilos de programación que usan los lenguajes de programación. Un lenguaje de programación puede usar más de un paradigma. Dependiendo del problema que se va a resolver un paradigma resultará más apropiado que otro.

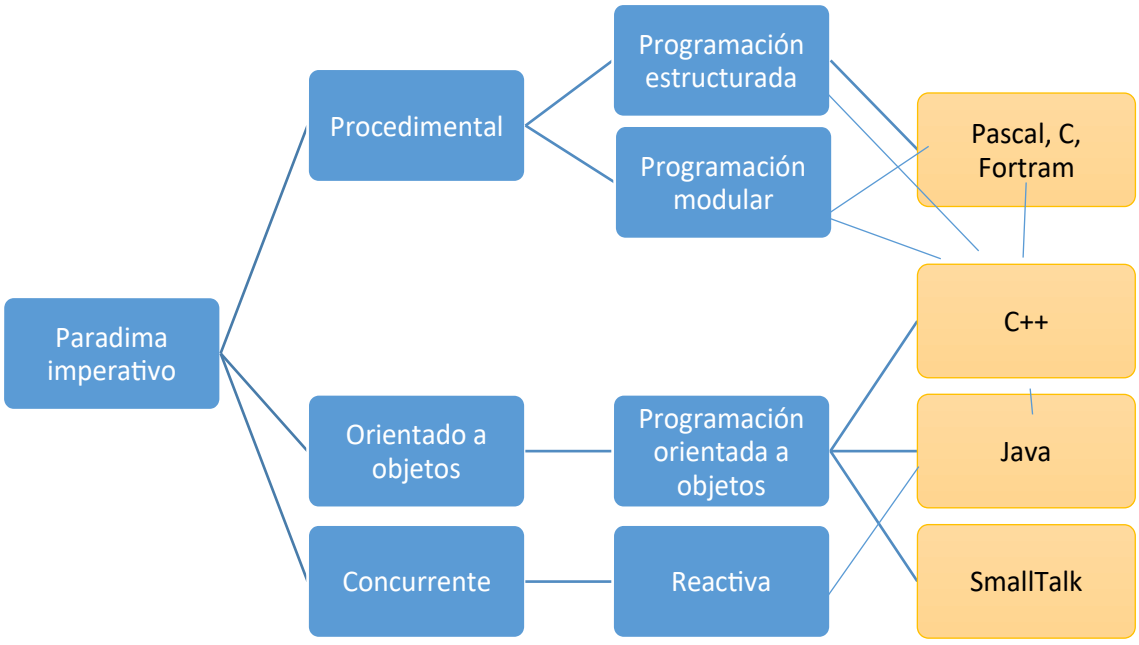
Hay que destacar que en la actualidad el uso del multiparadigma de programación está muy extendido. Este paradigma, como su nombre indica, se basa en hacer uso de varios paradigmas en un Proyecto, por ejemplo, hacer el desarrollo de la parte Backend con Programación Orientada a Objetos (POO) y la parte Frontend con Programación Reactiva con React JS de JavaScript.

Podemos definir 2 grandes paradigmas de programación:

a. Paradigma imperativo: se describe paso a paso un conjunto de instrucciones que deben ejecutarse para variar el estado del programa y hallar la solución, es decir, un algoritmo en el que se describen los pasos necesarios para solucionar el problema.

b. Paradigma declarativo: Se describe el problema que se quiere resolver, pero no las instrucciones necesarias para solucionarlo

La siguiente figura muestra la subclasificación del paradigma imperativo, que es el que interesa para este tema:



La programación estructurada, la programación modular, la orientada a objetos y la reactiva implementan el paradigma de la programación imperativa utilizando los lenguajes que se muestran en el gráfico.

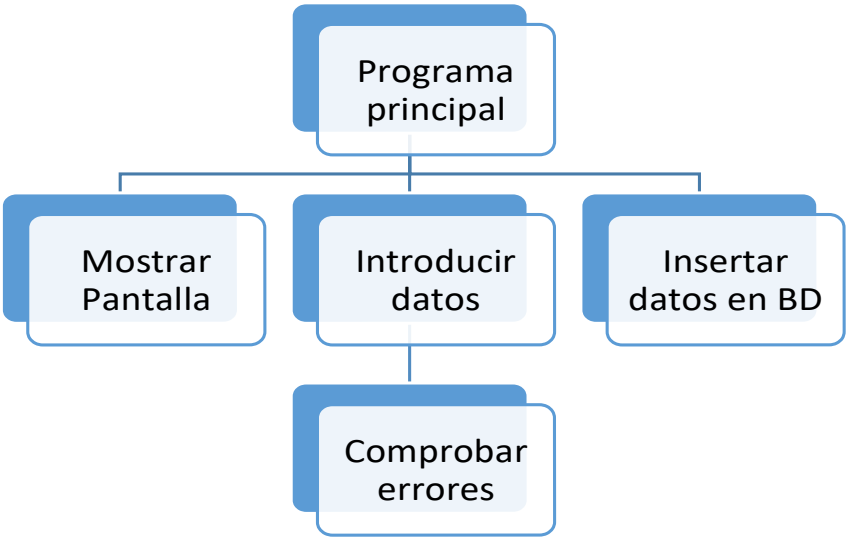
Lenguajes de programación estructurados y modulares

Son los lenguajes basados en la programación estructurada. Un programa estructurado es el que utiliza las tres construcciones lógicas: estructura secuencial, condicional y repetitiva.

El uso de estas 3 estructuras lógicas hace que los programas sean más fáciles de leer, es decir, puede ser leído secuencialmente desde el comienzo hasta el final sin perder la continuidad de loa que hace. El problema es que todo el código se concentra en un único bloque, si el programa es demasiado grande o el problema que se va a resolver es complejo, resulta difícil su lectura y manejo.

Normalmente, cuando se habla de programación modular, se refiere a la división del programa en partes más manejables conocidas como módulos. Así, un programa estructuras puede estar compuesto por un conjunto de estos módulos, cada uno tendrá una entrada y una salid. La comunicación entre ellos debe estar perfectamente controlada y se debe poder trabajar de forma independiente con cada uno de ellos.

A continuación, se muestra un programa que se divide en varios módulos: el módulo raíz o programa principal controla el resto de los módulos; el primero muestra una pantalla de entrada de datos, en el segundo módulo se hace la entrada y comprobación de los datos y en el tercero se insertan los datos en la BD.



A esta evolución de la programación estructurada se le llama **programación modular.** Esta división en módulos aporta una serie de ventajas.

- Al dividir el programa en varios módulos, varios programadores pueden trabajar simultáneamente
- Los módulos se pueden reutilizar en otras aplicaciones
- Es menos costoso resolver pequeños problemas de forma aislada que abordar el problema de forma global.

Ejemplos de lenguajes estructurados con programación modular son: Pascal, C o Fortran.

La programación modular se basa en la realización de una serie de descomposiciones sucesivas del algoritmo inicial, que describen el refinamiento

progresivo del repertorio de instrucciones que van a constituir el programa. Consiste en dividir un programa en módulos o subprogramas con el fin de hacerlo más legible y manejable. Un programa quedará formado por una serie de módulos, cada uno de los cuales realiza una parte concreta de la tarea total.

Se presenta históricamente como una evolución de la programación estructurada para solucionar problemas de programación más grandes y complejos de lo que ésta puede resolver.

Diagrama del funcionamiento de un subprograma.

Al aplicar la programación modular, un problema complejo debe ser dividido en varios subproblemas más simples, y estos a su vez en otros subproblemas más simples. Esto debe hacerse hasta obtener subproblemas lo suficientemente simples como para poder ser resueltos fácilmente con algún lenguaje de programación. Ésta técnica se llama refinamiento sucesivo, divide y vencerás ó análisis descendente (Top-Down).

La programación estructurada apareció a principios de los setenta de la mano de E.W. Dijkstra. Tiene las características de la programación modular y, además, establece que un programa debe cumplir las siguientes condiciones:

- El programa se desarrolla dividiéndolo en partes bien definidas y fácilmente ensamblables. Estas partes deberán poder ser modificadas de forma individual y sin que esto afecte al resto del programa.
- El código deberá contener la información suficiente para ser comprendido y verificado de forma autónoma
- En mayor o menor medida, el programa deberá contener una serie de elementos que son: estructuras básicas, recursos abstractos y razonamiento top-down.

La programación estructurada está enfocada al desarrollo del programa final y la programación modular es parte del diseño de programación y determina si un dato puede ser entrada o salida de otro proceso.

Un programa debe tener las siguientes características:

- Poseer un solo punto de entrada y uno de salida o fin para control del programa
- Existir caminos desde la entrada hasta la salida que se pueden seguir y que todos son recorridos en algún momento
- Todas las instrucciones son ejecutables y no existen lazos o bucles infinitos

2. FUNCIONES Y PROCEDIMIENTOS

2.1. Introducción

Los problemas reales que se plantean a un departamento de informática requieren programas de cierta complejidad y a veces de gran tamaño. Abordar el diseño de un programa de estas características de una forma directa es una tarea bastante difícil. Lo más adecuado es descomponer el problema, ya desde su fase de análisis, en partes cuya resolución sea más asequible. La programación de cada una de las partes se realiza independientemente de las otras, incluso, por diferentes personas.

El diseño descendente o top-down consiste en una serie de descomposiciones sucesivas del problema inicial, que describen el refinamiento progresivo del repertorio de instrucciones que van a formar parte del programa.

La utilización de esta técnica tiene los siguientes objetivos:

- Simplificación del problema y de los subprogramas resultantes de cada descomposición.
- Las diferentes partes del programa pueden ser programadas de modo independiente e incluso por diferentes personas
- El programa final queda estructurado en forma de bloques o módulos, lo que hace más sencilla su lectura y mantenimiento.

2.2. Programas y subprogramas

Un programa diseñado mediante esta técnica quedará constituido por dos partes claramente diferenciadas:

Programa principal que describe la solución completa del problema y consta de llamadas a subprogramas, de instrucciones primitivas y sentencias de control. Debe contener pocas líneas de código, y en él, ser verán los diferentes pasos del proceso que se ha de seguir para la obtención de los resultados deseados. Las llamadas a subprogramas son indicadores al procesador de que debe continuar la ejecución del programa en el subprograma llamado, regresando al punto de partida una vez lo haya concluido.

Subprogramas: Su estructura coincide con la de un programa por lo que puede tener sus propios subprogramas correspondientes a un refinamiento del mismo. El subprograma es invocado a través de su nombre y se le pueden pasar una seria de datos (parámetros) con los que podrá trabajar; así mismo, podrá devolver unos datos de salida.

La función del subprograma es resolver de modo independiente una parte del problema. Es importante que relace una función concreta en el contexto del problema; no obstante, a veces se convierte en subprograma un conjunto de instrucciones, cuando éstas se tendrían que repetir varias veces en diferentes lugares del programa. De esta manera, el conjunto de instrucciones que se van a repetir parece una sola vez en el listado del programa.

Un programa es ejecutado por el procesador sólo cuando es llamado por el programa principal o por otro subprograma.

2.3. Parámetros

Todo programa utiliza unos datos de entrada y produce unos resultados. Para esta labor se usan las variables de enlace o parámetros. Cada vez que se realiza una llamada a un subprograma, los datos de entrada son pasados por medio de determinadas variables y, análogamente, cuando termina la ejecución del subprograma, los resultado regresan mediante otras o las mismas variables .

Los parámetros pueden ser de dos tipos:

- Formales: Variables locales de un subprograma utilizadas para la recepción y el envío de los datos
- Actuales: Variables y datos enviados, en cada llamada de subprogra, por el programa o subprograma llamante.

Los formales son siempre fijos para cada subprograma, mientras que los actuales pueden ser cambiados para cada llamada. En cualquier caso, ha de haber una correspondencia entro los parámetros formales y actuales en su número, colocación y tipo.

Paso de parámetros

El proceso de emisión y recepción de datos y resultado mediante variables de enlace se denomina paso de parámetros. Puede realizarse de dos maneras diferentes:

- Paso por valor: Para suministrar datos de entrada al subprograma. Un parámetro pasado por valor es un dato, o una variable global que contienen un dato, de entrar para el subprograma. Esta variable no puede ser modificada por el subprograma que copia su valor en el parámetro forma correspondiente para poder utilizarlo
- Paso por referencia: Para entrada y salida o solo salida. Un parámetro pasado por referencia es una variable del programa o subprograma llamante , que puede contener o no un dato para el subprograma llamado, el cual coloca un resultado en esa variable que queda a disposición del llamante una vez concluida la ejecución del subprograma.

Existen dos tipos de subprogramas que son las funciones y los procedimientos

2.4. Funciones

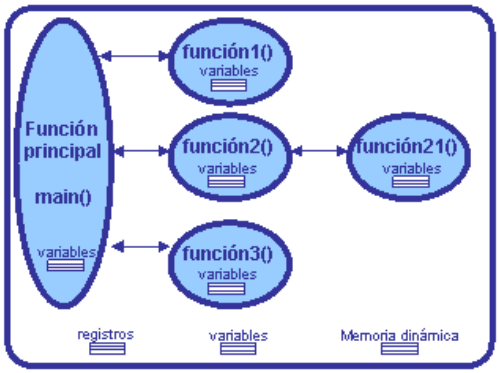
Son módulos que, recibiendo uno o varios datos de entrada, son capaces de producir datos de salida. Están divididas en dos partes:

- La cabecera o declaración de la función, constituida por el nombre de la función y los argumento entre paréntesis. Además, al igual que las variables, al principio o al final, aparecerá el tipo de datos que devuelve la función cuando se invocada.
- Su definición o cuerpo de la función. Donde se desarrolla el código que contiene la función con las instrucciones y declaración de variables necesarias.

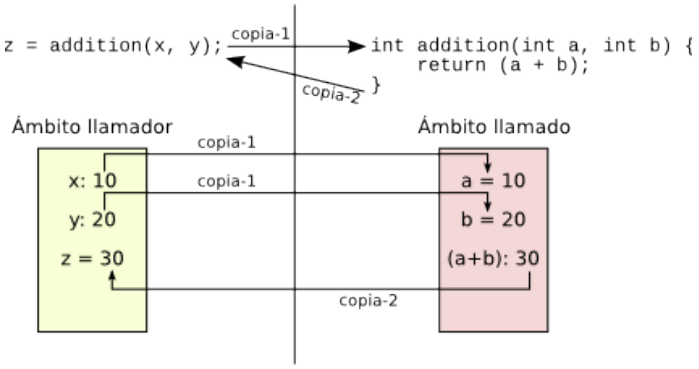
Existen dos tipos de funciones:

- Las internas, que son aquellas que vienen definidas en el lenguaje de programación y no es necesario ni declararlas ni definirlas ya que el programa ya lo ha hecho por nosotros. Estar funciones se almacenen en librerías y permiten ahorrar mucho tiempo al programador.
- Las externas, que son definidas por el programador y para las cuales, habrá que hacer su declaración y definición.

Una función se puede invocar desde cualquier parte del programa poniendo su nombre seguido de los argumentos, es decir, los valores que se quieren introducir como datos de entrada.



Por ejemplo:



2.5. Procedimientos

Aunque las funciones son herramientas de programación muy útiles para la resolución de problemas, su alcance es limitado. A menudo, se requieren subprogramas que calculen varios resultados en vez de uno solo o que realicen la ordenación de una serie de números, etc. En estas situaciones la función no es apropiada y se necesita disponer de otro tipo de subprograma llamado procedimiento o subrutina. La diferencia con las funciones es que no producen ningún dato de salida o producen varios.

```
public void intercambiar(Empleado x, Empleado y)
{
    Empleado temp = x;
    x = y;
    y = temp;
}
```

3. ACOPLAMIENTO Y COHESIÓN

El término "acoplamiento" hace alusión al grado de dependencia que tienen dos unidades de software.

Si hablamos de funciones, el acoplamiento nos da una idea de lo dependientes que son dos funciones entre sí. Es decir, en qué grado una función puede hacer su trabajo sin la otra. Si hablamos de librerías, el acoplamiento nos dará una idea de en qué medida el contenido de una librería puede hacer su trabajo sin la otra.

Cuando dos unidades de software son absolutamente independientes (cada una puede hacer su trabajo sin contar para nada con la otra), encontramos el grado más bajo de acoplamiento, y decimos que ambas unidades están totalmente desacopladas.

Nuestro objetivo al programar o diseñar debe ser el de tener un acoplamiento lo más bajo posible entre dos unidades de software cualesquiera. Por supuesto, es imposible lograr un desacoplamiento total entre las unidades. Sin embargo, manteniendo lo más bajo posible el acoplamiento lograremos que las distintas "piezas" de nuestro software funcionen sin depender demasiado unas de otras. Eso redunda en una mejora considerable en la detección y corrección de errores, en una mayor facilidad de mantenimiento y sobre todo, en la reutilización de esas "piezas" de software.

Por otro lado, la cohesión tiene que ver con la forma en la que agrupamos unidades de software en una unidad mayor. Por ejemplo, la forma en la que agrupamos funciones en una librería, o la forma en la que agrupamos métodos en una clase, o la forma en la que agrupamos clases en una librería, etc…

Se suele decir que cuanto más cohesionados estén los elementos agrupados, mejor. El criterio por el cual se agrupan es la cohesión.

El objetivo del programador debe ser maximizar la cohesión de los módulos reduciendo el acoplamiento de las funciones y procedimientos que lo integran.

4. RECURSIVIDAD

En informática, resolver un problema mediante recursividad significa que la solución depende de las soluciones de pequeñas instancias del mismo problema.

Niklaus Wirth, en su libro Algoritmos+E.D=Programas, dice:

"El poder de la recursión evidentemente se fundamenta en la posibilidad de definir un conjunto infinito de objetos con una declaración finita. Igualmente, un número infinito de operaciones computacionales puede describirse con un programa recursivo finito, incluso en el caso de que este programa no contenga repeticiones explícitas"

La recursividad es una técnica potente de programación que puede utilizarse en lugar de la iteración (bucles) para resolver determinado tipo de problemas. Consiste en permitir que un subprograma se llame a sí mismo para resolver una versión reducida del problema original. Muchas técnicas de programación avanzada como backtracking o algoritmos voraces utilizan la recursividad como desarrollar sus cálculos. Además, estructuras de datos populares como los árboles están construidos de manera recursiva y por tanto, para tratar con ellas, la recursividad es la técnica adecuada en términos de rendimiento y ahorro de recursos. De igual forma, multitud de objetos geométricos se pueden definir recursivamente como, por ejemplo, los fractales.

Frente a una determinada gana de problemas se puede optar por una solución iterativo (no recursiva) o una solución recursiva. Existen situaciones en las que el uso de la recursividad permita soluciones (programas) mucho más simples (y elegantes). No obstante, no convienen abusar de esta herramienta, pues podría dar lugar a resultados impredecibles y de difícil comprensión.

El uso de esta técnica es apropiado cuando el problema que se va a resolver o la estructura de datos que se va a procesar tiene una clara definición recursiva.

Por ejemplo, para calcular el factorial de un número n, entero positivo, se hará a partir de su definición:

0!=1;

n!=n*(n-1)*(n-2)*…*3*2*1, si n>0

Se dice que un subprograma es recursivo si entre sus instrucciones tiene una llamada a sí mismo. Por tanto, la definición recursiva de factorial sería la siguiente:

0!=1;

n!=n*(n-1)!, si n>0

```
int factorial(int n)
{
    if (n==1)
      return 1;
    else
      return n*factorial(n-1);
}
```

5. Bibliografía

- Metodología de la programación, Alcalde E.
- Manualweb.net
- https://www.arkaitzgarro.com/
- Aprenderaprogramar.com