**Unidad 1**

Lenguajes de programación

**Un lenguaje de programación es un sistema notacional** **para describir computaciones** en una forma legible tanto para la máquina como para el ser humano

**Objetivo:** Construir programas escritos por personas que serán comprendidos y ejecutados por una computadora.

**Cuestiones de Diseño**

* Legibilidad
* Abstracción: Permitir concentrarse en un problema a nivel general, dejando de lado los detalles irrelevantes.

**Tipos de Abstracciones**

* **Abstracción de datos:** Técnica de inventar nuevos tipos de datos que sean más adecuados a una aplicación.
* **Abstracciones de control:** Modificación de la secuencia de ejecución de un programa en una determinada situación. (Ejemplo: for, if, while, etc, en C).
* **Abstracciones procedimentales:** Es una secuencia nombrada de instrucciones que tienen una función específica y limitada. (Ejemplo: uso de funciones en C).

**Definición de un lenguaje de programación**

La definición de un lenguaje de programación necesita una descripción precisa y completa, además de un traductor que transforme el lenguaje de alto nivel en uno de bajo nivel adecuado para la ejecución.

**Partes de la definición:**

**Sintaxis:** Las reglas de sintaxis nos dicen cómo se escriben los enunciados, declaraciones y otras construcciones del lenguaje.

**Semántica**: Hace referencia al significado de esas construcciones.

**Ejecución de lenguajes de alto nivel en computadoras de bajo nivel**

**Simulación por software o Interpretación:** Es un software que recibe un programa en lenguaje de alto nivel, lo analiza y lo ejecuta.

**Traductor:** Es un “procesador de lenguajes” que acepta programas en cierto lenguaje fuente como entrada y produce programas funcionalmente equivalentes en otro lenguaje objeto.

De aquí salen los lenguajes interpretados y los compilados.

**Traductores:**

* Ensamblador.
* Compilador.
* Cargador o editor de vínculos.
* Preprocesador o macroprocesador.

**Esquema General de un Traductor**

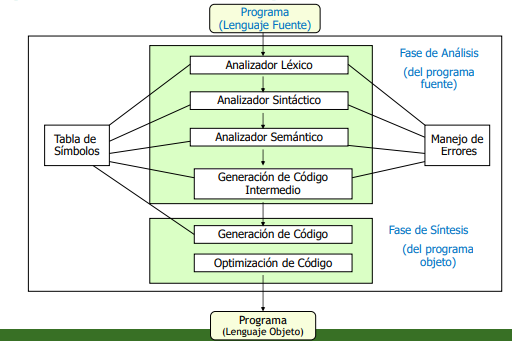
Traducción

Ejecución

**Compilación**

Un compilador es un traductor que transforma un programa escrito en un lenguaje de alto nivel (lenguaje fuente) en otro programa equivalente escrito en un lenguaje de bajo nivel.

**Esquema general de un compilador**



**Implementación de un lenguaje de programación**

Cuando se crea un lenguaje de programación, las decisiones sobre las estructuras de datos y los algoritmos que se utilizan afectan la forma en que ese programa se ejecuta.

Se deben tomar en consideración los recursos de hardware y software disponibles en la computadora y los costos de su uso.

**Principios a tener en cuenta:**

**Eficiencia**

**Eficiencia de código:** Permite que el traductor genere un código ejecutable eficiente.

**Eficiencia de traducción:**  Permite que el código fuente se traduzca con rapidez y con un traductor de tamaño razonable

**Eficiencia de implementación:** Eficiencia con la que se puede escribir un traductor.

**Eficiencia de la programación:** Facilidad para escribir programas en el lenguaje y darle mantenimiento al programa.

**Regularidad**

La regularidad indica lo bien que se integran las características del lenguaje.

**Generalidad**: Un lenguaje alcanza generalidad evitando casos especiales en cuanto a disponibilidad o uso de constructores.

**Ortogonalidad**: Se refiere al atributo de combinar varias características de un lenguaje de manera que la combinación tenga significado.

**Uniformidad:** Hace referencia a la consistencia de la apariencia y comportamiento de los constructores del lenguaje.

**Las no-uniformidades son de dos tipos:**

a) dos cosas similares que no parecen serlo, se comportan similarmente, o

b) cosas no similares, se comportan similarmente cuando no debieran.

**Paradigma de Programación**

Un paradigma de programación "consiste en un método para llevar a cabo cómputos y la forma en la que deben estructurarse y organizarse las tareas que debe realizar un programa".

**Clasificación de los Paradigmas**

En la **programación declarativa** se programa diciendo lo que se quiere resolver a nivel de usuario, pero no las instrucciones necesarias para solucionarlo. Esto último se realizará mediante mecanismos internos de inferencia de información a partir de la descripción realizada.

En la **programación imperativa** se describe paso a paso un conjunto de instrucciones que deben ejecutarse para variar el estado del programa y hallar la solución. De aquí derivan:

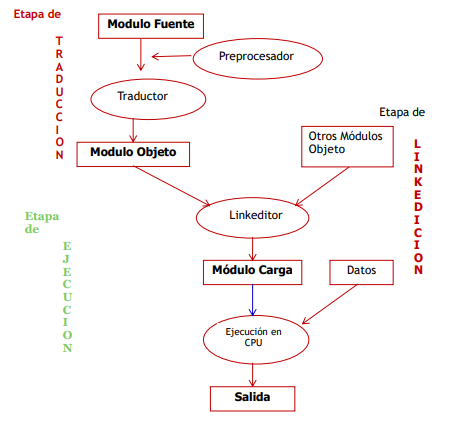
* **Programación orientada a objetos:** Encapsula elementos denominados objetos que incluyen tanto variables como funciones.
* **Programación dinámica:** Proceso de dividir problemas en partes pequeñas para analizarlos y resolverlos de forma cercana al óptimo. (divide y vencerás)
* **Programación orientada a eventos:** Tanto la estructura como la ejecución de los programas van determinados por los sucesos que ocurran en el sistema, definidos por el usuario o que ellos mismos provoquen.
* Un **lenguaje procedural o imperativo** es un lenguaje controlado por mandatos o instrucciones.

El proceso de ejecución de programas procedurales por la computadora, tiene lugar a través de una serie de estados, cada uno definido por el contenido de la memoria, los registros internos y los almacenes externos durante la ejecución.

* El almacenamiento inicial de estas áreas define el **estado inicial** de la computadora.
* Cada paso en la ejecución transforma este estado en otro nuevo, **transición de estado.**
* Cuando termina la ejecución del programa, **estado final** viene dado por el contenido final de las áreas antes mencionadas.

**Etapas en la obtención de programa ejecutable en lenguaje C**

Antes de ejecutar un programa en lenguaje C, existen ciertas etapas que se deben realizar, ya que todas ellas contribuyen a la obtención del programa ejecutable. Ellas son: **edición, compilación, enlace, carga y ejecución.**

****

**Unidad 2**

OBJETO DE DATOS

Representación de un elemento u objeto de la realidad de manera que pueda ser procesado por una computadora. Recipiente para almacenamiento y recuperar valores de datos.

**Tipos de datos**

**Objetos de datos definidos por el programador:** Creados y manipulados por el programador a través de declaraciones y enunciados. Ej: variables, arreglos, etc.

**Objetos de datos definidos por el sistema:** Se crean automáticamente sin intervención del programador Se construyen para mantenimiento durante la ejecución de un programa.

* Pilas de almacenamiento en tiempo de ejecución
* Registros de activación
* Memoria intermedia de archivos

**Atributos de los datos:**

* Identificador
* Tipo
* Valor
* Ubicación

**Clasificación:**

* Elemental
* Estructurado
* Variable: Su valor puede cambiar, pero no su nombre y su tipo
* Constante: Esta enlazado en forma permanente a un valor.

**Tiempo de vida**

**Durante la ejecución de un programa:**

* Algunos objetos existen desde el comienzo o pueden crearse dinámicamente durante la ejecución.
* Algunos objetos persisten durante toda la ejecución o se pueden destruir durante la misma.

**Enlace o Ligadura**

**Enlace o ligadura de un elemento de programa**

El enlace o ligadura se refiere a la conexión entre un elemento de programa (como una variable o una función) y una característica o propiedad específica en el código.

**Tiempo de enlace**

Momento del programa durante el cual se realiza el enlace

**Estáticas o tempranas:**

* Tiempo de definición del lenguaje
* Tiempo de implementación del lenguaje
* Tiempo de traducción

**Dinámicas**:

* Tiempo de ejecución

Los enlaces estáticos favorecen la eficiencia de ejecución mientras que los enlaces dinámicos favorecen la flexibilidad del lenguaje.

**Tabla de Símbolos**

Es una estructura de datos que contiene una entrada por cada identificador declarado encontrado en el programa fuente, con la siguiente información:

* Tipo de variable
* Tipo de parámetro formal
* Nombre del subprograma
* Entorno de referencia, etc.

**Tipos de datos**

Un tipo de datos es una clase de objetos de datos ligados a un conjunto de operaciones para crearlos y manipularlos

* **Primitivos:** Datos integrados al lenguaje
* **Definidos por el programador.**

**Especificación**

* **Atributos**: Distinguen objetos de ese tipo. Ej: Para tipo arreglo nombre dimensión, tipo de dato de componentes.
* **Valor:** Valores posibles para ese tipo de dato
* **Operaciones**

**Implementación**

* **Representación de almacenamiento:** Forma en que se representa en la memoria
* **Representación de algoritmos y procedimientos que definen las operaciones**

**Tipos elementales de datos**

Es una clase de objeto de datos que **contiene un solo valor**

**Atributos:** Se mantienen invariables

**Valor:** Generalmente ordenado, existe un máximo y un mínimo y dados dos valores distintos del dato, uno es mayor que el otro

**Operaciones**

* **Primitivas**: Se especifican como parte de la definición del lenguaje.
* **Definidas por el programador**: Procedimientos que el programador genera.

**Implementación de tipos elementales de datos**

**Representación de almacenamiento en memoria**

Hace referencia a la forma en que se representan los datos en la memoria, depende del lenguaje.

**Representación de algoritmos y procedimientos que definen operaciones**

1. Como operación de hardware
2. Simulada por software a través de subprogramas
3. Simulada por software como una secuencia de código de línea

**Declaraciones**

Una declaración **es un enunciado** del programador que sirve para comunicar al traductor información **que permite establecer ligaduras.**

**Ventajas**

* Informan al traductor acerca de la representación de almacenamiento para un objeto de datos
* Permiten que el traductor, determine cuál es la operación particular a la que hace referencia el operador, en el caso de operaciones polimórficas
* Informan sobre el tiempo de vida de un objeto
* Verificación estática de Tipos

**Verificación de tipos**

La verificación de tipos es el **proceso que realiza el compilador o el intérprete** para verificar que todas las construcciones en un programa tengan sentido en términos de los tipos de sus constantes, variables, procedimiento o cualquier otra entidad que involucre.

**Verificación estática**

**Se realiza durante la traducción** de un programa, con información proporcionada por el programador a través de las declaraciones.

Si un lenguaje detecta en forma estática los errores de tipo de un programa, se dice que es de tipo fuerte o fuertemente tipificado

**Pasos:**

1. El traductor recoge información (variables y operaciones) desde la TS la cual se va construyendo a partir de las de las declaraciones.
2. Verifica la validez de los argumentos y determina el tipo de datos del resultado, información que guarda para verificar operaciones posteriores.
3. Si la operación es homonimia o polimórfica, el nombre de la operación se puede reemplazar por el nombre de la operación específica que usan esos argumentos.
4. Como no se realizan marcas de tipo en los objetos en tiempo de ejecución, se gana almacenamiento y tiempo de ejecución.

**Verificación dinámica**

La verificación dinámica es realizada en **tiempo de ejecución por los intérpretes.**

**Pasos**:

1. Para implementarla se guarda una marca de tipo en cada objeto (descriptor), que indica el tipo de datos asociado al mismo. Cuando se realiza la verificación, lo primero que se efectúa es la verificación de marcas de tipo de argumentos.
2. La operación se realiza si los tipos de argumentos son correctos, de lo contrario se indica un error
3. Para cada operación, se anexan las marcas de tipo a sus resultados, para verificación de las operaciones siguientes.

**Verificación de tipos en C**

Los compiladores de C aplican **verificación estática** de tipos durante la traducción, muchas inconsistencias en tipos no causan errores de compilación, sino que son automáticamente eliminados, presentando o no un mensaje de advertencia.

**Conversión de tipos**

**Conversión Implícita (coerción)**

Son invocadas automáticamente en ciertos casos de discordancia. El principio básico que gobierna las coerciones es no perder información.

**Conversión Explícita**

El programador llama a funciones integradas para realizar una conversión de tipos. (En C se usa el operador cast)

**Tipos de datos enteros**

**Especificación:** Un objeto de datos de tipo entero tiene como atributo solo su tipo. Los valores que puede tomar un objeto de datos es un subconjunto ordenado de números enteros

**Implementación:** La implementación de un tipo de dato entero usa la representación de almacenamiento definida por el hardware y las operaciones primitivas relacionadas a enteros.

**Operaciones sobre enteros**

**Operaciones Aritméticas:** +, -, \*, /, ++, --

**Operaciones Relacionales**: <, >, <=, >=, ==, !=

**Operación de asignación**

**Rvalue**: Hace referencia al valor que contiene una variable

**Lvalue**: Se refiere a la ubicación en memoria de la variable

**Tipos de datos reales**

**Especificación:** Un objeto de este tipo puede tomar cualquier valor real dentro de una serie ordenada de valores, y limitada por un valor mínimo y un valor máximo determinado por el hardware.

**Implementación:** Norma IEEE 754 para representación de reales de punto flotante.

**Tipo de dato Booleano**

**Especificación**

Representa a objetos de datos que pueden tomar uno de dos valores posibles: True o False.

**Operaciones**:

* Conjunción
* Disyunción (inclusive)
* Negación
* Implicación (en algunos lenguajes)

**Implementación**

Dos maneras de representación dentro de la unidad de almacenamiento direccionables (un byte o una palabra):

1. Se usa un solo bit del byte para representar con 0 o 1 y los restantes se ignoran. (generalmente se usa el bit de signo)
2. Un valor 0 en toda la unidad de almacenamiento representa falso, cualquier otro valor representa verdadero.

**Tipo de dato carácter**

**Especificación:** Toma como valor un solo carácter. El conjunto de posibles valores se corresponde con el conjunto de caracteres que maneja el hardware y el Sistema Operativo subyacente.

Un conjunto de caracteres ordenados esta ordenado según el orden alfabético entre cadenas

**Implementación**: Son manejados por el hardware y el Sistema Operativo (S.O) subyacente, debido a su uso en la entrada y salida de información.

**Tipo de dato Apuntador o Puntero**

Un objeto del tipo apuntador, posee la dirección o localidad de otro objeto (valor L del objeto) o puede contener un apuntador nulo. No todos los lenguajes proveen este tipo de datos.

Tipos de datos elementales de lenguaje C para gestión dinámica de memoria.

**Permite:**

* Al programador, en tiempo de ejecución crear variables y destruirlas cuando las considere innecesarias
* Crear estructuras de datos dinámicas (por ejemplo listas enlazadas).
* Devolver resultados a través de los parámetros de una función.

**Punteros en C: Declaración y Operadores**

**Declaración de puntero:** < tipo > \* < nombre variable >

**Operador de dirección (&):** Permite obtener la dirección de memoria de una variable.

**Operador de indirección o de contenido (\*)**: Permite obtener el contenido de la posición de memoria apuntada por un puntero

**Tipos de datos estructurados**

Un objeto de datos que está construido por un agregado de otros objetos de datos, llamados componentes, se conoce como un objeto de datos estructurado o una estructura de datos.

**Definidos por el programador:** Registros, Arreglos, etc.

**Definidos por el sistema durante la ejecución del programa:** La pila que usa el sistema para trabajar con funciones.

**Especificación**

**Número de componentes**

* Estructuras de tamaño fijo: Número de componentes invariable durante su tiempo de vida (Arreglos y Registros).
* Estructura de tamaño variable: Número de componentes cambia durante su tiempo de vida. Usan generalmente un apuntador para vincular componentes (Listas, Archivos).

**Tipos de componentes**

* Estructuras homogéneas: componentes del mismo tipo (Arreglos).
* Estructuras heterogéneas: componentes de distinto tipo (Registros).

**Mecanismo de selección de una componente**

**Arre[i]** selección de la íesima componente.

**m.legajo** selección del campo legajo del registro.

**Organización de las componentes**

* Unidimensional o según una serie lineal (arreglo unidimensional, registro, cadena de caracteres, listas.)
* Multidimensional (arreglo multidimensional, registro cuyos componentes son registros, listas cuyos componentes son listas, etc.)

**Operaciones sobre estructuras de datos**

1. Operaciones particulares sobre las componentes de una estructura de datos (selección directa o secuencial)

Ejemplo: Arre[0] = 5;

1. Operaciones sobre una estructura de datos completa conjunto limitado de operaciones, dependiendo del lenguaje.  
   Ejemplo: Arre1 = Arre2;

**Representación de almacenamiento**

1. **Representación Secuencial:** Usa un solo bloque de memoria contigua, incluye tanto componentes como descriptor.
2. **Representación Vinculada:** Usa varios bloques de memoria no contiguos vinculados a través de punteros.

**Arreglos Unidimensionales – Especificación**

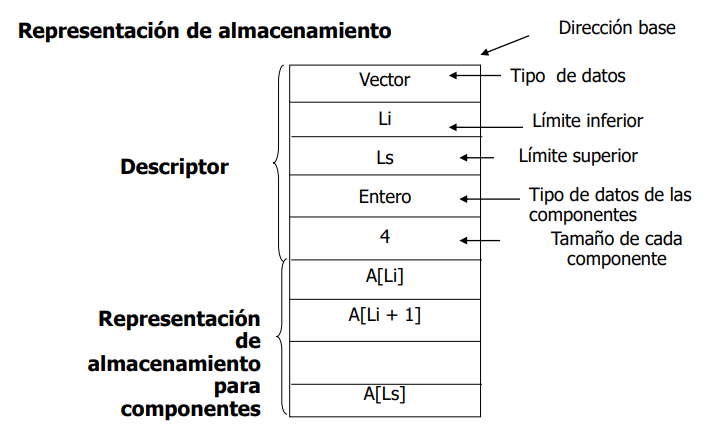
**Estructura homogénea de tamaño fijo** organizada como una serie lineal simple

**Atributos**

* Número de componentes
* Tipo de dato
* Subíndice para seleccionar una componente

**Operaciones sobre arreglos unidimensionales**

* **Selección de una componente:** subindización
* **Operación de Referenciamiento**
* **Operación de Selección**

****

**Arreglos Bidimensionales (Matriz)**

**Arreglo Bidimensional es un arreglo de arreglos.**

**Especificación**

En este caso para cada dimensión debe especificarse un intervalo de subíndices.

**En C**: int A[10][6]; arreglo bidimensional de 10 filas y 6 columnas.

**Representación**

En general, un arreglo de cualquier dimensión tiene representación en orden por filas. Esta representación implica que el arreglo se divide en subarreglos para cada elemento del intervalo del primer subíndice. Cada uno de estos subarreglos es un arreglo.

Para el caso de un arreglo bidimensional int t[10][30], es un arreglo que tiene 10 subarreglos de 30 componentes cada uno de ellos.

**Tamaño del arreglo en memoria:** Es el número de elementos por el tamaño (en bytes) de cada uno de ellos.

**Arreglos y Punteros en lenguaje C**

Existe una estrecha relación entre indexación en un arreglo y aritmética de punteros.

**Acceso a la dirección de un elemento de un arreglo:**

1. Nombre del elemento precedido por un &. Ejemplo: (&b[i]).
2. Nombre del arreglo más una cantidad entera i. Ejemplo: (b+i).

**Acceso a un elemento o contenido de esas direcciones**

**b[i] o \*(b+i)**

**CADENAS DE CARACTERES**

**Objeto de datos compuesto de caracteres.**

**Especificación y sintaxis**

* **Cadena de longitud fija declarada en el programa**. char nom\* +=“FIN”
* **Cadena de longitud variable hasta un límite máximo declarado en el programa**. char nom[20]
* **Cadena de longitud no determinada.** char \* nom

**CADENAS DE CARACTERES EN C**

* C no provee tipo de datos cadena, utiliza arreglos de caracteres.
* Las funciones de manejo de cadena estan en la librería <string.h>
* El carácter nulo ‘\0’ sigue al último carácter de la cadena, es anexado por el traductor al almacenar la cadena.

**REGISTROS**

**Especificación**

Estructura de datos compuesta por un número fijo de componentes de igual o distinto tipo, de longitud fija.

**Operación básica**: selección.

**Atributos de un registro**

* Número de componentes, campos o miembros
* Tipo de datos de cada componente
* Identificador para nombrar cada componente

**Implementación**

Se usa una representación secuencial de almacenamiento. La selección de una componente se implementa con facilidad pues los nombres de los campos se conocen en tiempo de traducción.

**Registros en Lenguaje C – Struct**

Declaración de variable :

struct <identificador> < var 1>,...,<var n>;

**Acceso a Componente:**

<identif\_struct>. <identif\_campo>

**Asignación de valores a campos o atributos de una estructura**

* Inicialización
* Asignación Directa
* Lectura desde teclado

En C también se pueden crear y utilizar arreglos de registros.

**Punteros a struct**

**p=&al;** apunto a la struct a través de p

**Acceso a campos**

* Puntero ->miembro
* (\*puntero).miembro

UNIDAD 3

FUNCIONES EN C

**Definición de Tipos**

La definición de tipos permite crear nuevos tipos de datos y operaciones sobre ese tipo, de modo que pueda ser usado como un tipo de dato provisto por el lenguaje.

**Typedef**

En varias ocasiones puede ser útil definir nuevos nombres para tipos de datos, estos nombres o 'alias' nos hace más fácil la declaración de variables y parámetros.

Para esto C dispone de la palabra clave typedef, cuyo formato es:

typedef < identificador>;

**Sistema de tipos**

Un sistema de tipos incluye los métodos utilizados para la construcción de tipos, el algoritmo de equivalencia de tipos, y las reglas de inferencia y corrección de tipos.

**Ventajas de la definición de tipos:**

* Simplificación la estructura del programa
* Modificación más eficiente.
* En el uso de subprogramas, facilita el pasaje de argumentos, pues evita repetir la descripción del tipo de datos.

**Subprogramas**

Son trozos de códigos utilizados para dividir un programa con el objetivo de que cada uno realice una tarea determinada para resolver una parte del problema

**Ventajas de usar subprogramas**

* Reusabilidad de tarea
* Mayor claridad y legibilidad al programa principal, aun cuando los subprogramas no se invoquen de manera repetida.

**Especificación de un subprograma**

* **Nombre**
* **Prototipo o signatura**: número de argumentos, su orden, y tipo de cada uno.
* **Cuerpo del subprograma**: descripción de la tarea que realiza o función que calcula

**Implementación de un subprograma**

La implementación está definida por el cuerpo del subprograma

* Declaraciones de datos locales
* Acciones

**Definición, invocación y activación de subprogramas**

**Definición de un subprograma:** Es una propiedad estática de lenguaje; en **tiempo de traducción** es la única información disponible.

**La traducción de la definición de un subprograma es una plantilla que permite generar activaciones en tiempo de ejecución.**

La plantilla se divide en partes con el fin de ahorrar memoria: **segmento de código y registro de activación**

El **segmento de código** es la parte estática compuesta por las constantes y el código ejecutable generado a partir de **enunciados del cuerpo de la función.** (invariable durante la ejecución)

**Prólogo:** Bloque de código que el traductor introduce al comienzo del segmento de código. Permite tareas de creación del registro de activación, transmisión de parámetros, creación de vínculos y actividades similares de mantenimiento.

**Epílogo:** Conjunto de instrucciones que el traductor inserta al final del bloque de código ejecutable, para realizar acciones que permitan devolver resultados y liberar almacenamiento destinado al registro de activación.

**Activación de un subprograma:** se genera en tiempo de ejecución cuando se lo llama o invoca. Al terminar la ejecución, la activación se destruye.

El **registro de activación** es la parte dinámica, compuesta por:

* Parámetros formales
* Resultados de la función
* Datos locales
* Punto de retorno
* Áreas temporales de almacenamiento
* Vinculaciones para referencias de variables no locales

**Los registros de activación se crean cada vez que se invoca un subprograma y destruyen cada vez que el mismo concluye con un retorno.**

**Funciones en C**

Una función es un conjunto de sentencias que realiza una determinada tarea, que retorna como resultado cero o un valor.

Un programa en C está formado por una o más funciones, siendo main la función principal por donde se inicia la ejecución del programa.

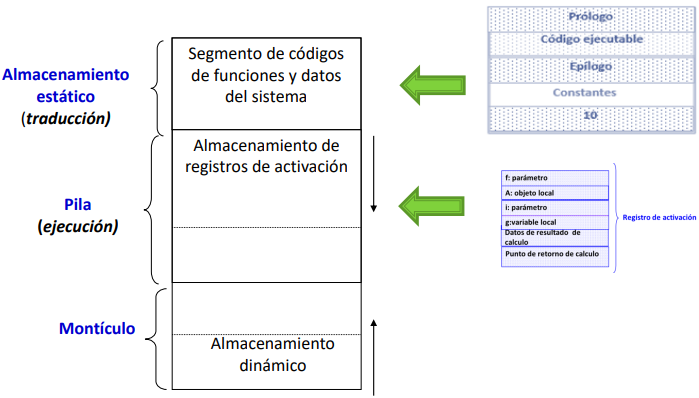
Al invocarla desde algún punto del programa, se ejecutan las sentencias que forman parte de ella.

Al finalizar su ejecución, se devuelve el control al punto desde donde fue invocada.

**Funciones en C – Parámetros**

* Los parámetros actuales deben coincidir en tipo, orden y cantidad con los parámetros formales.
* El cuerpo de la función debe incluir al menos una sentencia return para devolver cero o un valor
* La sentencia return permite que se devuelva el control al punto de llamada o invocación.

**Organización de la memoria en C**



**Pasaje de Parámetros**

Un programa se comunica con sus funciones a través de los parámetros. Existen distintas formas de pasar parámetros a una función.

**Pasaje por valor**

* Los parámetros formales se consideran como variables locales que toman como valor inicial el valor de los parámetros actuales.
* Los parámetros formales pueden cambiar sus valores a través de asignaciones sin que estos cambios afecten los valores de los parámetros actuales.
* **Desventaja:** Se produce duplicación del área de memoria.

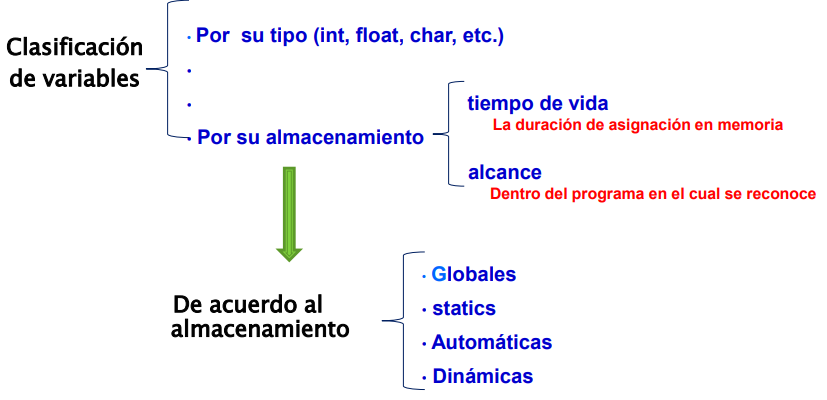
**Pasaje por dirección**

* Consiste en el paso por valor de una dirección, por ello puede usarse para cambiar el contenido de la memoria apuntada por ese puntero.
* El pasaje de parámetros por dirección permite retornar más de un resultado desde la función.
* En C, para pasar un parámetro por dirección, se utiliza el operador de dirección &, al momento de invocar a la función. Luego, el operador de indirección \* debe utilizarse en la función para acceder al valor almacenado en esa dirección.

**Pasaje por referencia**

* Para realizar un pasaje por referencia el parámetro actual a pasar debe ser una variable con una dirección asignada.
* El pasaje por referencia pasa la ubicación de la variable, por lo que el parámetro formal se transforma en un alias del parámetro actual de modo que cualquier cambio que se realiza en el parámetro formal se siente en el parámetro actual.
* Esto puede interpretarse como que a una misma área de memoria se asignan dos nombres distintos.

**Arreglos como parámetros de funciones**

Cuando se envía un arreglo como parámetro de una función, el parámetro formal toma la dirección de memoria de la primer componente del arreglo.****

**Variables automáticas**

Son las declaradas en la lista de parámetros o en el cuerpo de una función.

* Se almacenan en el registro de activación de la función
* Su alcance está restringido a la función en la que se declaran

**Variables externas**

* **Se definen una sola vez, fuera de cualquier función y pueden ser inicializadas.**
* **Su alcance no se restringe a una función**, todas las funciones pueden tener acceso a ella a través de su nombre.
* **Se almacenan en el área de almacenamiento estático.**
* En algunos casos, para el uso de una variable externa en una función es necesario el uso del especificador externo.

**Variables estáticas**

* Son locales a una función, se declaran con el especificador static, pueden ser inicializadas.
* Su alcance se restringe a la función en la que se declaran.
* Se almacenan en el área de almacenamiento estático, por ello mantienen la información a lo largo de la ejecución.

**Bloque en lenguaje C**

Un bloque es una **secuencia de declaraciones**, seguidas por una **secuencia de enunciados**, rodeados por **marcadores sintácticos** (llaves de inicio-terminación). *Es todo el código dentro de un par de llaves de cierre y apertura {}.*

**Alcance de un vínculo en lenguaje C**

Las declaraciones vinculan atributos a un nombre. Alcance de este vínculo es la región del programa donde este vínculo se mantiene. En C, lenguaje estructurado en bloques, el alcance de un vínculo queda limitado al bloque donde aparece la declaración asociada, y a los bloques contenidos en el interior.

**Tabla de Símbolos**

La tabla de símbolos es una estructura de datos, se crea en tiempo de traducción, permite dar apoyo a la inserción, búsqueda y cancelación de identificadores con sus atributos asociados. Se utilizan operaciones SET (activar un bloque) y RESET (desactivar un bloque) para gestionar el alcance de los identificadores en bloques de código anidados

La estructura de datos que soporta una Tabla de Símbolos puede ser una tabla, una pila, un árbol, etc.

Los símbolos se guardan en la tabla con su nombre y una serie de atributos opcionales que dependen del lenguaje y objetivos del procesador. Entre ellos se encuentra:

* Nombre de identificador.
* Dirección a partir de la cual se almacenará la variable en tiempo de ejecución, dirección a partir de la cual se colocará el código en caso de funciones.
* Tipo del identificador. Si es una función, el tipo del resultado.
* Tipo de los parámetros de las funciones o procedimientos.

UNIDAD 4

RECURSIVIDAD

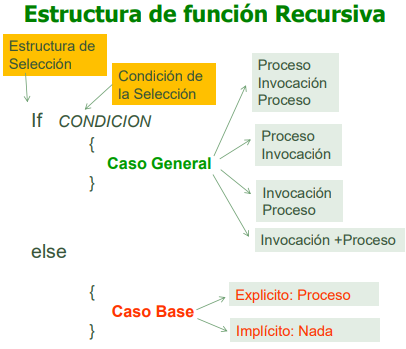
**Es una alternativa a la iteración, un proceso mediante el cual se puede definir una función en términos de sí misma.**

**Construcción y ejecución de funciones recursivas**

Al construir una función recursiva, se debe asegurar la existencia de una estructura de selección que permita identificar:

**Caso base** (puede haber más de uno): Permite detener la invocación sucesiva de la función (no la vuelve a invocar). Es un conjunto de sentencias que se ejecutan cuando la recursión termina.

**Caso general** (puede haber más de uno): permite que la función se invoque a sí misma con valores de parámetros que cambian en cada llamada; acercándose cada vez más al caso base.

****

En la ejecución, las sentencias que aparecen después de cada invocación no se resuelven inmediatamente, **éstas quedan pendientes.**

Si una función recursiva tiene **variables locales**, se creará un conjunto diferente de estas variables para cada llamada recursiva.

Cada invocación recursiva, genera un registro de activación y se apila en el stack o pila. Esos registros de activación se van desapilando en el orden inverso a como fueron apilados.

**Ventajas de la Recursividad:** La recursividad puede hacer que el código sea más claro y conciso al expresar soluciones de manera elegante para problemas complejos. En algunos casos, la recursividad es la única forma natural de abordar un problema

**Desventajas de la Recursividad:** La recursión puede tener un mayor uso de memoria debido a la pila de llamadas que se utiliza para almacenar las llamadas recursivas pendientes.

**Recursión versus Iteración**

Ambas se basan en una estructura de control.

* La iteración utiliza una estructura de repetición
* La recursión utiliza una estructura de selección.

Ambas implican repetición:

* La iteración utiliza la repetición de manera explícita
* La recursión consigue la repetición mediante repetidas llamadas a una misma función.

Ambas involucran una prueba de terminación:

* La iteración termina cuando falla la condición de continuación del ciclo
* La recursión termina cuando se reconoce un caso base.

**Unidad 5**

Estructuras Dinámicas

Pueden **crearse en tiempo de ejecución** y su espacio de almacenamiento pueda **liberarse** cuando no se las necesite **y modificar su tamaño** durante la ejecución del programa.

**Almacenamiento Estático vs Almacenamiento Dinámico**

Estáticas o de Almacenamiento Fijo:

* El sistema asigna el área que ocupará, a partir de su declaración
* El almacenamiento permanece fijo durante la ejecución de la función que la declara
* No se pueden utilizar esas posiciones de memoria para otros fines durante la ejecución

Dinámicas:

* **Se crean en tiempo de ejecución**, en cualquier punto, no sólo al entrar a un subprograma, **a petición**
* Se puede liberar su espacio de almacenamiento cuando no se las necesite
* • En el caso de estructuras, se puede modificar su tamaño durante la ejecución del programa.

**Montículo o heap**

**Operaciones:**

* **Alojamiento: Se demanda un bloque para almacenar un objeto** de cierto tamaño. **Resultado de la función es un puntero** a un trozo contiguo de memoria dentro del montículo.
* **Desalojo:** Se indica que ya no es necesario conservar un objeto, la memoria que ocupa queda libre para ser reutilizada

Las operaciones dependen del tipo de lenguaje, y pueden ser:

* **Explícitas** (imperativo)**:** el programador invoca operaciones, incluye en código fuente una instrucción que demanda cierta cantidad de memoria para la ubicación de un dato.
* **Implícitas** (lógico y funcional)**:** el sistema realiza las operaciones, la memoria requerida es calculada por el compilador en función del tipo correspondiente al objeto que se desea alojar.

**Características del lenguaje para definir variables dinámicas**

**Poseer un tipo elemental de datos apuntador** (o puntero), llamado también tipo de referencia o de acceso.

Una **operación de creación de objetos** de tamaño fijo, que devuelve el valor L del bloque de almacenamiento creado y ese valor L se convierte en el valor r de una variable del tipo apuntador.

Una **operación de desreferenciar** para objetos del tipo apuntador, que permite acceder al valor del objeto al cual apunta.

**Características e implementación de punteros**

Los apuntadores pueden referenciar objetos de un único tipo de datos (verificación estática de tipos) o referenciar a cualquier tipo de datos (verificación dinámica).

**Implementación:**

* **Direcciones absolutas:** El apuntador contiene la dirección de memoria real del bloque de almacenamiento del objeto.
* **Direcciones relativas:** El apuntador contiene el desplazamiento del objeto respecto a la dirección base de un bloque de almacenamiento más grande.

**Manejo del Montículo en lenguaje C**

**Función malloc:** Solicita a sistema operativo un bloque de memoria en el montículo, del tamaño especificado en el argumento.

El bloque es de tipo void, por lo que se debe hacer un casteo al tipo de dato utilizado. Si el resultado es NULL, no había espacio disponible.

**Función free:** Libera bloques asignados previamente por malloc.

**Algunos Inconvenientes que pueden surgir:**

* **Presencia de elemento basura**: Cuando habría espacio disponible para nuevo uso, pero dicho espacio no está en la lista de espacios libres, por ello se ha vuelto inaccesible.
* **Referencias desactivadas:** Esto sucede cuando se libera el espacio asignado a un objeto, pero no se destruyen todas las rutas de acceso.

**Sistema Operativo y Gestión de Asignación de Memoria**

El Sistema Operativo maneja una tabla de direcciones de memoria que indican el espacio de memoria ocupado (para evitar que otro programa modifique datos que no le corresponden)

Cuando se usa malloc, el sistema operativo registra en su tabla de direcciones de memoria que un bloque ha sido ocupado, y envía a malloc la dirección de inicio de ese bloque.

Cuando se usa free , el Sistema Operativo registra en su tabla de direcciones de memoria que dicho bloque ha sido desocupado.

Arreglos Dinámicos

En C, un arreglo de 10 componentes se puede declarar de dos formas: int b[10] ó int \*b.

Teniendo en cuenta que se debe realizar la asignación de memoria para el segundo caso.

b=(int \*) malloc (sizeof(int)\*10);

Listas

Una lista es un conjunto de elementos del mismo tipo que puede crecer o contraerse sin restricciones. Todos los elementos son accesibles y se puede insertar y suprimir un elemento en cualquier posición de la lista.

**Implementación**

* Mediante arreglos: Listas secuenciales
* Mediante Punteros: Listas enlazadas

**Características de Listas secuenciales**

* Se debe conocer cantidad de elementos o asignar suficiente memoria si no se conoce la cantidad.
* Los elementos se almacenan en celdas contiguas de memoria.
* Sus elementos se acceden de manera directa.
* Insertar un elemento en cualquier lugar de la lista, obliga al desplazamiento de una posición a todos los elementos que le siguen.
* La eliminación de un elemento, excepto el último, requiere desplazamientos para llenar el vacío formado.

Listas implementadas mediante punteros: Listas enlazadas

Cada elemento deberá contener además de su información intrínseca, un puntero a otro elemento de estructura similar, en el caso del último elemento, su puntero contendrá el valor NULL.

**Característica de las Listas enlazadas**

En esta representación cada elemento (nodo) de una lista debe tener dos partes:

* Una contiene la información de cualquier tipo de dato simple o estructurado (excepto File)
* La otra parte, contiene un enlace o puntero que indica la posición del siguiente elemento.
* Los elementos se pueden almacenar en celdas de memoria no contiguas

**Ventaja:**

* Desaparece el problema de los costosos desplazamientos de elementos para insertar y eliminar componentes de la lista en cualquier posición.
* No se necesita conocer a priori la cantidad de elementos de la lista.

**Desventaja**:

* Búsquedas Secuenciales.
* Requiere un espacio adicional para los apuntadores.

**Unidad 6**

Archivos

Estructura compuesta por datos almacenados en forma organizada en un dispositivo de almacenamiento secundario: discos rígidos, CD, pendrive, DVD

**Ventajas del uso de archivos:**

* Datos pueden ser usados por distintos programas y en distintos procesos.
* Capacidad de almacenamiento de memoria secundaria superior a memoria principal.
* El uso de archivos permite tener en memoria principal sólo aquella parte de datos que necesita ser accedida por el programa en un momento dado.
* El tamaño de los archivos está limitado sólo por la cantidad de memoria secundaria disponible.

**Desventaja**: El acceso a los datos de un archivo es más lento que los datos residentes en memoria principal.

**Organización de Archivos**

* **Organización Secuencial:** Las componentes se ubican en posiciones físicas contiguas en el soporte de almacenamiento
* **Organización Directa:** Las componentes ocupan posiciones físicas relacionadas mediante una clave, que puede ser un dato de la componente o el resultado de un cálculo.

**Criterios de clasificación de archivos**

Por la dirección del flujo de datos

* Archivos de entrada.
* Archivos de salida.
* Archivos de entrada/salida.

Según el tipo de valores permitidos a cada byte.

* Archivos de texto (con formato).
* Archivos binarios (sin formato).

Según el tipo de acceso

* Secuencial
* Directa o aleatoria

Según la longitud de registro que se almacena.

* Archivos de longitud variable
* Archivos de longitud constante

Archivos organizados secuencialmente con acceso secuencial

**Declaración de un archivo en el lenguaje C**

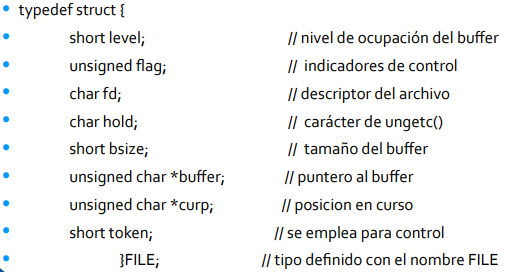
FILE \*archi;

Con esta declaración, en tiempo de compilación se habilita un espacio de memoria para una estructura de tipo FILE, que queda referenciada a través de la variable declarada.

La definición de esta estructura depende del compilador, pero en general mantiene información sobre:

* La posición actual de lectura/escritura
* El estado del archivo
* Un puntero a un buffer, área de memoria que sirve de almacenamiento intermedio entre el archivo físico y el programa que lo maneja.

Estructura FILE



En realidad, una variable de tipo FILE \* **representa un flujo de datos** que se asocia con un dispositivo físico de entrada/salida (el archivo “real” estará almacenado en disco).

**Operaciones básicas en el manejo de archivos**

**Creación y/o apertura:** Se crea un manejador lógico o vínculo entre la variable tipo archivo y el archivo físico.

(En C) fopen: vincula una variable del programa con el archivo físico en el dispositivo externo

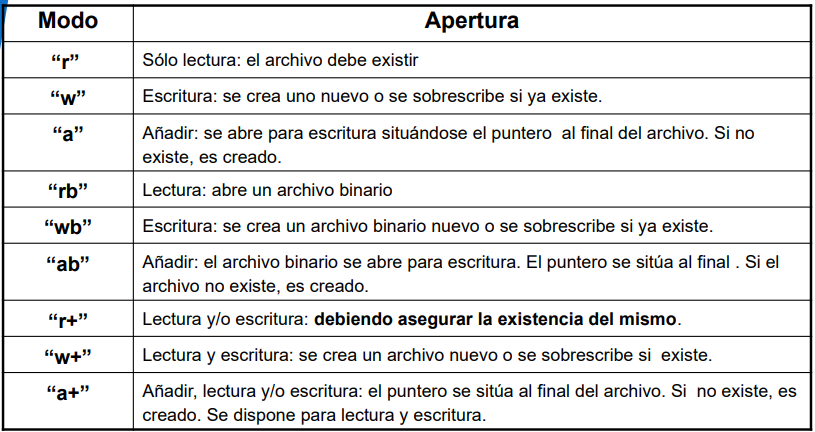
FILE \*archi ;

archi = fopen(“Datos.dat”, “wb”);

Parámetros:

* **nombre:** cadena de caracteres que contiene un nombre de archivo físico.
* **modo:** especifica modo de acceso en que se va utilizar el archivo, lectura, escritura o para agregar información, y tipo de valores permitidos a cada byte, de texto (t) o binario (b).

Resultado: puntero a FILE si la operación es exitosa, NULL caso contrario



**Grabar información:** Consiste en guardar una componente en el archivo físico abierto.

**Recuperar información:** Abrir archivo y leer componente.

**Modificar información:** Consiste en alterar una componente en un archivo físico.

**Eliminar información:** Proceso de quitar componentes de un archivo.

**Cerrar:** Libera el vínculo que une el programa a través de la variable tipo archivo, con el archivo físico vinculado.

(En C) fclose(FILE \*archivo);

**Parámetro:** un puntero a la estructura FILE del archivo que se quiere cerrar, NULL si se quiere cerrar todos los archivos abiertos.

**Resultado:** cero indica que el archivo ha sido correctamente cerrado, caso contrario indica que hubo un error.

**Archivos organizados secuencialmente**

**Pueden accederse:** secuencialmente o en forma directa

**Características de los archivos secuenciales:**

* Escritura de nuevos datos al final del archivo
* La lectura se realiza desde la primera componente.
* Solo se puede abrir en un modo a la vez (lectura o escritura)

**Archivos de Texto**

**Archivos de caracteres**

Los archivos de caracteres son una secuencia de caracteres almacenados en un dispositivo de almacenamiento secundario e identificado por un nombre de archivo.

Se modifican con fgets, fputs, fprintf, fscanf.

**Archivos sin formato**

Estos archivos son tratados como un bloque de datos binarios (en bytes) que tan solo tienen significado si se conoce la estructura: distribución y tipos de datos

Se modifican con fwrite, fread y feof.

**Archivos organizados secuencialmente con acceso directo**

El archivo tiene en las componentes un elemento llamado clave, que está en relación con la posición física del almacenamiento. Esto permite la ubicación precisa de la componente en el archivo.

Cuando se accede en forma directa a un archivo de acceso directo, hay que tener en cuenta algunas características:

* La escritura de nuevos datos se realiza al final del archivo sin que se pierda la relación de la clave con la posición física de la componente.
* Para leer una componente concreta se utiliza la clave para ubicar dicha componente

**Función fseek:** Esta función sirve para ubicar el puntero en el archivo, para lectura o escritura en el lugar deseado.

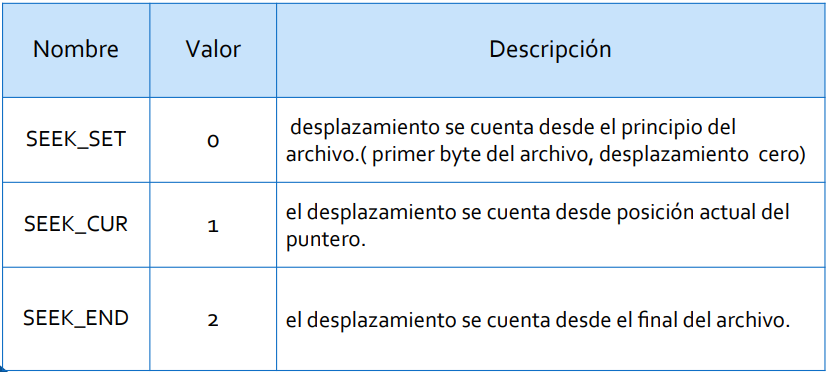
int fseek(FILE \*archivo, long int desplazamiento, int origen);

**Parámetros:**

* puntero a FILE del archivo que se quiere acceder
* valor del desplazamiento, expresado en bytes (positivo o negativo).
* punto de origen desde el que se calculará el desplazamiento.

**Valor de retorno:** cero si fue exitosa la función, distinto de cero si hubo error.

Posibles valores del parámetro origen:



**Función fgetpos:** Esta función se utiliza para obtener la posición actual del puntero del archivo. Dicha posición está dada en bytes

int fgetpos (FILE \*archivo, fpos\_t \*posicion);

**Parámetros:**

* Puntero a FILE
* La ubicación del registro dentro del archivo expresada en bytes.

**Valor de retorno:** cero si fue exitosa la función, distinto de cero si hubo error.

**fpos\_t** es un tipo de dato definido por el lenguaje que se utiliza para hacer referencia a las posiciones de los punteros de archivos.

**Eliminar información del archivo**

Para eliminar registros:

1. Ubicarse en el registro que se desea eliminar o buscarlo
2. Una vez que la componente está en memoria, se marca
3. Grabar componente marcada
4. Crear un archivo auxiliar con registros sin marcas
5. Borrar el archivo original
6. Renombrar el archivo auxiliar

**Función rename:** Cambia el nombre a un archivo físico de un dispositivo de almacenamiento

**int rename(const char\* viejo\_nombre, const char\* nuevo\_nombre);**

**Resultados:** cero si la operación fue exitosa, cualquier otro valor si hubo error.

**Función remove:** Elimina un archivo físico de un dispositivo de almacenamiento.

**int remove(const char \*nombre\_archivo);**

**Resultados:** cero si la operación fue exitosa, cualquier otro valor si hubo error.