UNIDAD 2

Ventajas que justifican el uso de C:

- Poderoso y flexible, con órdenes, operaciones y funciones de biblioteca que se pueden utilizar para escribir la mayoría de los programas que corren en la computadora.
- Se puede utilizar para desarrollar sistemas operativos, compiladores, sistemas de tiempo real y aplicaciones de comunicaciones.
- La existencia de numerosas bibliotecas que soportan aplicaciones de bases de datos, gráficos, edición de texto, comunicaciones, etc.

Estructura General de un programa en C:

1. Directivas de preprocesador
2. Declaraciones Globales
3. Función main()
4. Funciones definidas por el usuario
5. Comentarios del programa

Directivas de preprocesador: Las directivas son instrucciones que se dan al compilador antes de que el programa se compile. Todas las directivas de preprocesador comienzan con el signo #. Dos tipos de directivas que utilizamos hasta ahora son:

`#define` Para definir macros
`#include` Para incluir archivos definidos por el usuario o estándar.

Declaraciones Globales: Se sitúan antes de la función main(), indican al compilador que las funciones definidas por el usuario o variables declaradas pueden ser utilizadas desde cualquier parte del programa y desde cualquier función. Pueden tornar complicado el control de errores en el programa.

Función `main()`: Cada programa en C tiene y debe tener (obligatoriamente) sólo una función `main()`, que es el punto de entrada al programa.

Funciones definidas por el usuario: Se invocan por su nombre y el listado de parámetros actuales que puedan tener. Una vez que una función es invocada, el control del programa pasa a dicha función, desde la cual se ejecutarán las sentencias definidas en el cuerpo de la misma. Cuando la función finaliza, el control del programa retorna a la función llamadora.

Comentarios: Los comentarios estandar comienzan con la secuencia /* y terminan con la secuencia */. Todo texto situado entre las dos secuencias es un comentario. Los comentarios son ignorados por el compilador.

Elementos de un programa en C:

- Identificadores
- Palabras reservadas
- Signos de puntuación y separadores
- Comentarios

Identificador: Un identificador es una secuencia de caracteres, letras, dígitos y subrayados. Se utiliza para dar nombre a constantes, variables, tipos de datos, funciones, etc.
Reglas para formación de identificadores:

1. Debe ser una secuencia de letras, dígitos o subrayados.
2. Los identificadores son sensibles a mayúsculas
3. Pueden tener cualquier longitud
4. No pueden ser palabras reservadas

Palabras reservadas: Son palabras del lenguaje de programación asociadas con algún significado especial.

- Tienen un significado predefinido
- No pueden ser utilizadas como identificadores, nombre de variables o de funciones
- Son propias del lenguaje que se está utilizando

Signos de puntuación y separadores: Son signos especiales que permiten al compilador separar y reconocer las diferentes unidades sintácticas del lenguaje.

Comentarios: Sirven para documentación interna.

DATOS Y SU REPRESENTACIÓN

Los datos nos sirven para llevar registro de hechos del mundo real, utilizados para representar información, la cual implica que los datos estén procesados de manera útil y significativa para quien lo recibe.

Los datos se definen por tipos, un tipo de dato es un patrón que determina el conjunto de valores que pueden tomar las variables asociadas. También define la representación interna que tendrá y las operaciones definidas para dicho tipo.

Métodos de representación (de datos tipo entero):

1. Binario Puro: Consiste en expresar el dato en el sistema de numeración de base 2.
2. Módulo y Signo: Consiste en disponer un bit llamado bit de signo para representar el signo del número entero y los restantes bits representan el módulo del número.
3. Complemento a 1: Utiliza el bit más significativo para el signo del número y los bits restantes para el módulo, utilizando el binario puro para positivos y el complemento a 1 para negativos.
4. Complemento a 2: Utiliza el bit más significativo para el signo del número y los bits restantes para el módulo, utilizando el binario puro para positivos y el complemento a 2 para negativos, la ventaja del método es que existe una única representación para el cero.

Método de representación para float:

- Notación Exponencial o científica: El número se representa como el producto entre la mantisa y la base del exponenciación.

TIPOS DE DATOS EN C

Todos los tipos de datos simples o básicos de C son números. Los tres tipos de datos básicos son: enteros, números de coma flotante (reales) y caracteres. Estos tipos de datos

son atómicos o simples, lo que significa que las variables asociadas a estos tipos de datos permiten contener un solo dato por vez.

El tipo `void` es utilizado como tipo de dato asociado a aquellas funciones que no retornan valor en su nombre (para crear procedimientos). También cuando una función o procedimiento no requiere de parámetros para crear punteros genéricos en asignación dinámica de memoria.

MODIFICADORES DE TIPOS:
Son utilizados para alterar el significado o rango de los tipos de datos básicos.

- ➔ De Longitud:
  - ◆ `short`: Aplicable a enteros, indica entero corto.
  - ◆ `long`: Aplicable a enteros, duplica el rango del tipo porque duplica la cantidad de bytes.
- ➔ De Signo:
  - ◆ `signed`: Es el tipo por defecto.
  - ◆ `unsigned`: Enteros sin signo.

VARIABLES

Una variable es una posición de memoria representada por un identificador, que tiene asociado un tipo de dato y almacena un dato del tipo asociado, cuyo valor puede cambiar durante la ejecución del programa.
Características:

- Se declaran antes de ser utilizadas.
- Están asociadas a un tipo de dato que está asociado a un identificador en la memoria RAM.
- Pueden ser globales cuando se declaran antes de la función `main()` o locales cuando se declaran dentro de una función.

ATRIBUTOS DE UNA VARIABLE: Nombre, dirección, tipo de valor, tiempo de vida y ámbito.

PUNTEROS

Cuando en C se declara un puntero, se crea una dirección de memoria de un byte de tamaño. El dato que contiene esta variable es imprecisa, por lo que el puntero apunta a alguna dirección de memoria no determinada. Esto puede provocar el acceso a posiciones de memoria indeseadas. Para evitar esta situación, los punteros siempre y en todo momento deben contener direcciones de memorias válidas.

Reglas por las cuales se rigen los punteros:

- Es una variable como cualquier otra.
- Contiene una dirección que apunta a otra dirección en memoria.
- En esa posición se almacenan los datos a los que apunta el puntero.

CONSTANTES

En C existen 4 tipos de constantes:

- Constantes Literales: son los valores que se utilizan tal cual en el programa, se clasifican también en 4 grupos constantes enteras, reales, cadenas y caracteres.

- Constantes Definidas: reciben un nombre mediante la directiva #define, la cual es una directiva de preprocesamiento y no una sentencia, por lo cual no finaliza en punto y coma.
- Constantes Enumeradas: son utilizadas para crear listas de elementos. Cuando se procesa la sentencia el compilador asigna un valor secuencial a cada elemento, comenzando con el valor 0 para el primer elemento. Función `enum`
- Constantes Declaradas: El calificador `cons` permite declarar constantes asociando un tipo de dato a la misma.

UNIDAD 3

¿Qué es la modularización?

Es un método que consiste en dividir un problema complejo y/o de gran tamaño en problemas más pequeños.
La resolución de un problema comienza con una descomposición y nuevas descomposiciones, hasta que el problema original queda reducido a un conjunto de actividades básicas, que no se pueden o no conviene descomponer. Luego cada módulo se resuelve y las sub-soluciones se fusionan en la solución del problema original.

Ventajas de la modularización:

- Producir programas más fáciles de escribir y mantener
- Crear módulos independientes por sus parámetros
- Realizar pruebas independientes
- Reutilizacion de codigo
- Programas más limpios y con menos líneas de código

Concepto de Módulo: Es cada una de las partes de un programa que resuelve uno de los subproblemas en que se divide el problema original. Cada uno de los módulos tiene una tarea bien definida, además algunos necesitan de otros para poder operar. En este caso, los módulos pueden comunicarse mediante una interfaz de comunicación que debe estar bien definid'

CLASIFICACIÓN

Los módulos básicos se clasifican en procedimientos y funciones. Toda función o procedimiento realiza una tarea concreta y bien definida. La diferencia entre una función y procedimientos es el tipo de retorno que tienen, las funciones retornan un valor en el nombre, mientras que los procedimientos no, pues el nombre de la función tiene asociado un espacio de memoria capaz de contener un dato, mientras que el nombre del procedimiento es solo una referencia a las sentencias vinculadas al módulo y no tiene la capacidad de contener datos.

FUNCION

Tipo de retorno: Es tipo de dato que devuelve una función y normalmente aparece antes del identificador de la función. Puede ser cualquier tipo de dato excepto una función, una cadena de caracteres o un array. El tipo de retorno SIEMPRE debe coincidir con el valor devuelto por la función. Si no se especifica un tipo de retorno, se asume que es un entero.
Nombre de la función: Identificador.
Lista de parámetros: Es una lista de parámetros tipificados, cada declaración de parámetro indica tanto el tipo de dato del parámetro como su nombre.
Cuerpo de la función: Se encuentra entre llaves { }.
Declaración Local: Las constantes, variables y tipos de datos declarados dentro de la función son locales a la misma y no perduran fuera de ella.
Valor devuelto por la función: Una función devuelve un único valor, el cual debe coincidir con el tipo de retorno. El resultado se muestra con una sentencia return que generalmente se escribe al final de la función.

PROCEDIMIENTO

Tipo de retorno: No retorna resultados en su nombre.
Nombre del procedimiento: Identificador o nombre del procedimiento.
Lista de parámetros: Es una lista de parámetros tipificados, cada declaración de parámetro indica tanto el tipo de dato del parámetro como su nombre, debe coincidir también en cantidad.
Cuerpo del procedimiento: Se encierra entre llaves { }.
Declaración Local: Las variables, constantes y tipos de datos declarados dentro del cuerpo del procedimiento no perduraran fuera de él, pues son locales al mismo.
Valor devuelto por el procedimiento: Los procedimientos devuelven cero, uno o más valores, pero lo hacen en los parámetros que les son pasados por referencia.

MÓDULOS DEFINIDOS POR EL PROGRAMADOR
Podemos diseñar e implementar nuestros propios modelos, los cuales pueden formar parte de nuestros programas o de una librería no estándar.

ÁMBITO DE IDENTIFICADORES
Cada identificador tiene un campo de acción (ámbito), solo dentro de este campo de acción es posible utilizarlo.
Existen 4 tipos de ámbitos:

- Programa
- Archivo Fuente
- Funcion
- Bloque

Se puede asignar una variable para que esté asociada a uno de estos ámbitos. Tal variable es invisible fuera de su ámbito y solo se puede acceder a ella en su ámbito.
Existen modificadores de tipos o clases de almacenamiento que permiten modificar el ámbito y la permanencia de una variable dentro del programa, estos son:

- `auto`: Una variable se considera automática porque cuando se accede a la función se le asigna espacio en la memoria automática y se libera dicho espacio tan pronto finaliza la ejecución de la función. La utilización de la palabra auto es opcional, ya que es el modificador por defecto.
- `extern`: Se usa cuando una variable que se definió en otro módulo, se quiere utilizar en una función actual. Advierte al compilador que la variable ya se creó en otro módulo, por lo que no tiene que crearla, sino simplemente usarla.
- `static`: Cuando se antepone el modificador static a una variable
  - Local: Pasa de ser dinámica a ser estática. Su valor es recordado incluso si la función donde está definida finaliza su ejecución y se la vuelve a invocar más tarde.
  - Global: Se modifica su alcance y pasa de ser global a ser sólo accesible paralelas funciones del archivo en el que se la crea.
- `register`: Indica al compilador que almacene la variable en un registro de la máquina, que es el lugar más eficiente para guardar las variables. Esto se hace porque el trabajo con los registros del procesador es mucho más rápido que el trabajo con la memoria central. `register` sólo se puede aplicar a variables locales y a los parámetros formales de una función. Son ideales para el control de bucles.

VARIABLES LOCALES Y GLOBALES

Cada función puede definir sus propias variables locales definiendolas en su cuerpo, sin embargo C también permite definir variables fuera del cuerpo de cada función, éstas las conocemos como variables globales.

- Variables locales: Son aquellas que se declaran tanto dentro de la función main como de las demás funciones, su alcance está limitado solamente a la función en la cual están definidas.
- Variables globales: Están definidas fuera de cualquier función y pueden ser utilizadas por cualquier parte del programa. Solo puede inicializar una variable global en su definición. El valor inicial que se le asigne a la variable global debe expresarse como una constante y no como una expresión. Es posible darle el mismo nombre a una variable local y global en un mismo programa. En este caso el módulo no podrá utilizar la variable global, ya que le da preferencia a las locales sobre las globales.

TRANSFERENCIA DE INFORMACIÓN A/Y DESDE PROCEDIMIENTOS:

La comunicación entre funciones y el programa principal o bien entre las mismas funciones se lleva a cabo entre variables globales, parámetro por valor y parámetros por referencia.

PARÁMETROS ACTUALES	PARÁMETROS FORMALES
Estos se incluyen en las sentencias de llamada a una función. “Actual” se refiere a los valores o a las variables que intervienen como parámetros en cada llamada a la función. Los parámetros actuales deben coincidir en cantidad, tipo de datos y orden con los parámetros formales.	Estos se incluyen en la declaración. “Formal” hace mención a la forma que toman dichos parámetros, y el referirse a la forma indica la cantidad de parámetros, el tipo de datos asociados a cada uno de ellos, y el orden en que se presentan dichos parámetros.
PARÁMETROS POR VALOR	PARÁMETROS POR REFERENCIA
El paso de parámetros por valor consiste en copiar el contenido del parámetro actual que se quiere pasar en una variable dentro del ámbito local de un módulo. Se tendrán dos valores duplicados e independientes, con lo que la modificación de uno no afecta al otro.	El paso de parámetros pasados por referencia consiste en proporcionar al módulo invocado la dirección de memoria donde se aloja el dato. Si la función que lo recibe lo modifica, la variable original también se verá afectada. Estos parámetros se realizan con punteros.

UNIDAD 4

TIPOS DE DATOS:

- Simples: Se vinculan a variables que pueden contener solo un tipo de dato a la vez.
- Estructurados: Permiten registrar varios datos a la vez.

ARREGLOS UNIDIMENSIONALES

Un arreglo unidimensional es una colección finita, homogénea y ordenada de datos, en la que se hace referencia a cada elemento del arreglo por medio de un índice que indica su ubicación en el arreglo.
ARREGLOS UNIDIMENSIONALES COMO PARÁMETROS: Un vector completo pasa a una función mediante un parámetro que contiene la primera dirección de memoria asignada a la estructura, es decir que es un parámetro pasado por referencia. La referencia a un vector o a la dirección inicial, se especifica mediantes su nombre, sin corchetes ni subíndices.

ARRAYS Y PUNTEROS
Hay una fuerte relación entre arrays y punteros, sin embargo no son equivalentes. Declarar un puntero reserva memoria solo para el puntero. La variable puntero no está inicializada para apuntar a ningún espacio existente, por lo que inicialmente tiene cualquier valor no válido.

ARREGLOS BIDIMENSIONALES

Un arreglo bidimensional es una colección finita, homogénea y ordenada de datos, en las que se hace referencia a cada elemento del arreglo por medio de dos índices. El primero corresponde a la fila y el segundo a la columna.

TIPO DE DATO CADENA

En C una cadena se define como un array de caracteres (char) que termina en un carácter nulo, al declarar una cadena se debe considerar el espacio para el carácter nulo al final de la cadena.
Como una cadena es un array de caracteres, se puede usar como parámetro pasado por referencia exclusivamente.

ENTRADA Y SALIDA DE CADENAS CON FORMATO

Para leer una palabra podemos hacer uso de la función scanf() que nos permite leer una secuencia de caracteres no blancos, usando la marca de formato %s.
Para mostrar una cadena, utilizamos la función printf() con la marca de formato %s.

ENTRADA Y SALIDA DE CADENAS SIN FORMATO

Las funciones principales que realizan la entrada y salida sin formato son puts() y gets().
La función puts() escribe su argumento de tipo cadena, en la salida estándar, stdout, y reemplaza el carácter nulo de terminación de la cadena (\0) por el carácter nueva línea (‘\n’), provocando un salto de línea.
La función gets() tiene el inconveniente de que, al ser una lectura sin formato, puede provocar desbordamientos de memoria, ya que toma todos los caracteres hasta el fin de la cadena, pero si la variable de tipo cadena no tiene reservado espacio suficiente para la cantidad de caracteres ingresados, desborda la memoria, pudiendo afectar otras variables.

ESTRUCTURAS (REGISTROS)

Las estructuras son un tipo de dato estructurado, se utilizan para resolver problemas que involucran tipos de datos heterogéneos.
Una estructura es una colección de elementos finita y heterogénea. Cada componente de una estructura se denomina campo y cada uno de ellos puede ser simple o estructurado.

ACCESO A ESTRUCTURAS

Podemos acceder a una estructura usando el operador punto (.) o usando el operador puntero (->).

ESTRUCTURAS COMO PARÁMETRO

Pueden usarse las estructuras como parámetros pasados por valor o por referencia. Para pasar por referencia debe usarse el operador &.

- Cuando la estructura es pasada por referencia, el parámetro formal contiene la dirección de memoria de la estructura, es un puntero a la misma, por lo que el acceso a cada campo se realizará mediante el operador ->.
- Cuando la estructura es pasada por valor, el parámetro formal contiene los datos de la estructura, por lo que el acceso a cada campo se realiza mediante el operador . (punto).

UNIONES

- Es una colección de elementos finita y heterogénea en la que solo uno de sus componentes puede recibir valor a la vez.
- Son similares a las estructuras, pero se diferencian de ellas en que sus miembros comparten el mismo espacio de almacenamiento en la memoria.
- Su espacio de memoria reservado corresponde a la capacidad del campo de mayor tamaño.

UNIDAD 5

TIPO ABSTRACTO DE DATO

Definiciones:
TIPO DE DATO: Es un atributo de los datos que impone ciertas restricciones sobre los mismos, como los valores que pueden tomar y el tipo de operaciones que se pueden realizar sobre ellos.
ESTRUCTURA DE DATOS: Es una forma de organizar un conjunto de datos elementales con el objetivo de facilitar su manipulación.
TIPO ABSTRACTO DE DATO: **Es un modelo matemático con una serie de operaciones definidas para ese modelo.**

- Las operaciones que manipulan los datos están incluidas en las especificaciones del TAD, estas deben ser cerradas, es decir, sólo se debe acceder a los datos mediante las operaciones definidas sobre ellos.
- Para representar un modelo matemático básico de un TAD se emplean las estructuras de datos, quizá de tipos distintos, conectadas entre sí de diversas formas. No existe ninguna regla sobre qué operaciones debe manejar un TAD, esta es una decisión propia del diseño.
- La comunicación entre el programa de aplicación que utiliza el TAD y la implementación del mismo debe producirse únicamente a través de la interfaz que aportan las operaciones definidas en el TAD.

CARACTERÍSTICAS

- Las operaciones son cerradas, es decir, sólo se puede acceder a los datos mediante las operaciones definidas sobre ellos.
- Su comunicación con el programa de aplicación (programa cliente o programa principal) debe producirse únicamente a través de la interfaz que aportan las operaciones definidas en el TAD.
- La manipulación directa de los datos almacenados en la implementación del TAD no está permitida.

VENTAJAS DEL USO DE TADs

- Permite una mejor conceptualización y modelados del mundo real.
- Mejora la robustez del sistema.
- Mejora el rendimiento ya que permite la optimización de tiempo de compilación.
- Separa la implementación de la especificación.
- Permite la extensibilidad del sistema.
- Recoge mejor la semántica del tipo. Los TADs agrupan las operaciones y la representación de atributos.

ABSTRACCIÓN: Es la capacidad para encapsular y aislar información del diseño y la ejecución. Consiste en ocultar las características de un objeto y obviarlas, de manera que solamente se utilice el nombre del objeto en nuestro programa principal.
TIPOS DE ABSTRACCIÓN:

- ABSTRACCIÓN FUNCIONAL: Crear módulos e invocarlos mediante un nombre donde se destaca qué hace el módulo y se ignora cómo lo hace.
- ABSTRACCIÓN DE DATOS:

    - TIPOS DE DATOS: Proporcionados por los lenguajes de alto nivel. La representación usada es invisible al programador, al cual sólo se le permite ver las operaciones predefinidas para cada tipo.
    - TIPOS DEFINIDOS: Por el programador que posibilitan la definición de valores de datos más cercanos al problema que se pretende resolver.
    - TAD: Para la definición y representación de tipos de datos (Estructura de datos + operaciones).
    - OBJETOS: Son TADs a los que se les añade propiedades de reutilización y de herencia de código.

ENCAPSULAMIENTO: Este concepto consiste en el ocultamiento del estado y los datos de un TAD, de forma que solo es posible modificar los mismos mediante las operaciones definidas para dicho TAD. Además el usuario del TAD no se debe preocupar de cómo están implementadas las operaciones y atributos (datos), concentrado solamente en cómo debe usarlos.

VENTAJAS DEL ENCAPSULAMIENTO

- Facilidad de modificación.
- Posibilidad de reutilización.
- La dificultad inherente a la modificación de la implementación de un TAD es independiente del tamaño total del sistema.
- La simplicidad en el uso del TAD al ocultar los detalles de su funcionamiento y presentarlo en términos de sus operaciones.
- Constituye un mecanismo de integridad. Es decir, la dispersión de un fallo a través de todo el sistema es menor.
- Permite la sustitución de TAD con la misma interfaz y diferente implementación.

INTERFAZ E IMPLEMENTACION

A la estructura interna de un TAD se la denomina implementación y a la parte visible la que se presenta al exterior, interfaz.

TAD.c —> IMPLEMENTACIÓN
TAD.h —> INTERFAZ

La interfaz sirve como cubierta a la correspondiente implementación. Los usuarios de TAD tienen que preocuparse por la interfaz pero no por la implementación.
La solidez de un TAD reposa en la idea de que la implementación está escondida al usuario.

Solo la interfaz es pública, la interfaz se refiere al límite entre la implementación y los programas clientes. Una interfaz bien diseñada debe cumplir con los siguientes requisitos:

- Unificación: Escogidas de acuerdo a cierto tema, el enfoque se debe mantener.
- Simplicidad: En cuanto al número de parámetros, nombre adecuado.
- Generalidad: Resolver un buen grupo de posibilidades.
- Estabilidad: Se pueden realizar cambios en la librería/TAD pero no necesariamente en el programa cliente.

En una interfaz se pueden incluir:

- Prototipos de funciones.
- Declaraciones de constantes.
- Declaraciones de nuevos tipos de datos.

NUNCA SE INCLUIRÁ LA IMPLEMENTACIÓN EN LA INTERFAZ.

DIFERENCIAS ENTRE LIBRERIAS DINAMICAS Y ESTATICAS

- Las librerías estáticas quedan incluidas en el ejecutable, mientras que las dinámicas son ficheros externos, con lo que el tamaño de la aplicación es mayor cuando utilizamos librerías estáticas.
- Las librerías dinámicas son ficheros independientes que pueden ser invocados desde cualquier ejecutable (archivo binario), de modo que su funcionalidad puede ser compartida por varios ejecutables. Esto significa que solo se necesita una copia de cada fichero de librería (DLL) en el sistema.
- Si se realizan modificaciones en los módulos de una librería estática, es necesario recompilar todos los ejecutables que la utilizan, mientras que no es necesario en el caso de una librería dinámica, siempre que su interfaz se mantenga.
- Es más difícil la depuración y mantenimiento de aplicaciones que utilizan librerías dinámicas que las estáticas.
- Durante la ejecución de un archivo binario, las librerías estáticas que hubiesen intervenido en su construcción no necesitan estar presentes,en cambio las dinámicas deben estar en el mismo directorio o en el camino de búsqueda “PATH”.
- Las librerías estáticas solo se utilizan en la fase de construcción del ejecutable.Las dinámicas se utilizan durante la ejecución.
- Los ejecutables que utilizan las librerías estáticas solo incorporan los módulos de aquellas que se necesiten para resolver sus símbolos externos. Por el contrario, las librerías dinámicas deben ser cargadas en su totalidad (no son divisibles).
- Las librerías estáticas que se incluyen en un programa deben ser cargadas con el proceso de carga de dicho programa, mientras que las librerías dinámicas no necesariamente tienen que cargarse con la carga inicial, estas pueden ser cargadas en el momento que se necesita de su funcionalidad e incluso pueden ser descargadas cuando no resultan necesarias.
- El mecanismo de enlazado estático depende del compilador, mientras que el enlazado dinámico depende del sistema operativo.

UNIDAD 6

RECURSIVIDAD

La recursividad es una técnica de programación muy potente que puede ser utilizada en lugar de la iteración.
Para diseñar una solución recursiva de un problema se procederá a dividir el problema en instancias más pequeñas (sub-problemas), estos se resuelven de manera independiente y luego se combinan las soluciones parciales parciales para obtener la solución del problema original.

LOS PROCESOS RECURSIVOS SE COMPONEN DE:

- Al menos una invocación a sí mismo (llamada recursiva), en donde el valor de los parámetros cambiará en cada llamada, volviéndolo un caso más sencillo que el anterior y aproximándose en cada llamada al caso base.
- Una condición de terminación o salida, conocida como CASO BASE, que es la condición que no produce auto-invocación y evita los ciclos infinitos.

CADA LLAMADA RECURSIVA GENERA UNA NUEVA ZONA DE MEMORIA EN LA PILA.

TIPOS DE RECURSIVIDAD:

1. Indirecta (Mutua): Es el caso en donde una función A llama a otra función B, que a su vez vuelve a llamar a la función A.
2. Directa: Es el caso en donde la función A se invoca a sí misma, una o más veces directamente. Este tipo de recursividad se divide en 3 tipos:
    - DIRECTA MÚLTIPLE: Es el caso en donde la función A se invoca a sí misma más de una vez dentro de una misma instancia.
    - DIRECTA LINEAL: La función A se invoca a sí misma como mucho una vez dentro de una misma instancia,sin embargo esta invocación no es la última acción que realiza la función.
    - DIRECTA LINEAL FINAL: La función A se invoca a sí misma como mucho una vez dentro de una misma instancia y esta llamada recursiva es la última acción que efectúa como parte del cuerpo de su definición.

CONCEPTO DE EFICIENCIA

Un algoritmo es eficiente si realiza una administración correcta de los recursos del sistema en el cual se ejecuta.
Se puede analizar la eficiencia de un algoritmo desde dos aspectos:

- Tiempo de ejecución: Se consideran más eficientes aquellos algoritmos que cumplen con la especificación del problema en menor tiempo de ejecución.
- Administración o uso de la memoria: Serán eficientes aquellos algoritmos que utilicen las estructuras de datos adecuadas de manera de minimizar la memoria ocupada.

FORMAS DE ESTIMAR EL TIEMPO DE EJECUCIÓN DE UN ALGORITMO:

- Realizar un análisis teórico, calculando el número de asignaciones y de comparaciones que realiza.

- Realizar un análisis empírico, aplicando distintos juegos de datos a la implementación del algoritmo, midiendo su tiempo de respuesta. Este análisis es fácil de implementar pero es afectado por algunos factores como: La velocidad de la máquina y/o Los datos empleados en la ejecución.

COMPARACIÓN DE EFICIENCIA EN MÉTODOS ITERATIVOS VS RECURSIVOS

Si bien la recursión es una técnica potente de programación para resolver, mediantes soluciones simples y claras, problemas de gran complejidad, también tiene algunas desventajas desde el punto de vista de la eficiencia.

Factores que contribuyen a esto:

- Una simple llamada puede generar un gran número de llamadas recursivas.
- Puede clarificar programas pero no compensa la sobrecarga adicional.

UNIDAD 7

TIPOS DE DATOS DINÁMICOS: PUNTEROS

Un puntero en C indica en dónde encontrar determinados datos dentro de la memoria, es decir, contiene la dirección de memoria en donde se almacena un dato en un determinado momento de la ejecución de un programa.

Una variable de tipo puntero tiene asociados los mismos atributos que cualquier otra variable (nombre, tipo de dato asociado, dirección de memoria, valor almacenado, tiempo de vida y ámbito).
Los punteros se rigen por dos reglas básicas:

- Un puntero es una variable como cualquier otra.
- Una variable puntero contiene una dirección que apunta a otra posición en memoria, en esa posición se almacenan los datos a los que apunta el puntero.

DECLARACION E INICIALIZACION DE PUNTEROS

Al declarar un puntero se debe especificar el tipo de dato al que apunta la variable puntera. Una vez declarado el puntero debe realizarse su inicialización de manera correcta, pues cuando declaramos un puntero este puede contener basura de memoria, es decir que está apuntando a una posición de memoria desconocida, la cual puede pertenecera la memoria que se le asignó al programa o puede ser externa al programa.

ASIGNACIÓN DE UNA DIRECCIÓN DE MEMORIA VÁLIDA A UNA VARIABLE PUNTERO

1. Apuntarlo a una dirección de memoria ya reservada para el programa (apunta a una variable estática).
2. Reservar una zona de memoria específica para la variable puntero y hacer que apunte a ella. Para esto se utilizaran las funciones malloc y free de la biblioteca stdlib.h

DESCOMPOSICIÓN DE LA MEMORIA RAM

- PILA (stack) es una zona de memoria que se asigna de forma exclusiva a la aplicación, su tamaño es estático, es decir que no crece ni decrece durante la ejecución del programa.
- MONTÍCULO (heap) esta parte de la memoria es compartida y la administra el sistema operativo, que a petición de una aplicación, asigna de forma dinámica, por lo tanto varía al momento de la ejecución del programa. La mayor parte de la memoria que utiliza una aplicación proviene del montículo, y es aquí donde se van creando y destruyendo variables dinámicas.
- El código y los datos estáticos ocupan otra parte de la memoria asignada, el código es el programa en sí, la colección de instrucciones que hay que ejecutar y los datos estáticos son la información fija del programa. Esta información suele ubicarse por

encima de la pila, para evitar invasiones, o bien al otro extremo de la memoria, por la misma razón.

LISTAS ENLAZADAS

Este tipo de listas ajustan su consumo de memoria al tamaño de la secuencia de datos que almacenan, contrariamente al uso de las variables indizadas, que requieren de una reserva previa de memoria y por lo tanto, una reserva de memoria superior a la requerida por los datos.

Listas simplemente enlazadas: Es una secuencia de elementos (registros) dispuestos uno detrás de otro, en la que cada elemento se conecta al siguiente por un enlace o puntero.
Cada elemento (nodo) se compone de dos partes (campos):

- Primer Campo: Contiene la información y es un valor de tipo genérico (dato).
- Segundo Campo: Es un puntero que apunta al siguiente elemento.

Para saber dónde está la primera entrada de una lista, se guarda en una variable de la memoria, la dirección de la primera entrada. Esta variable apunta al principio de la lista y se suele llamar puntero inicial, inicio o cabeza,