**Conceptos básicos**

**Informática**: Estudia el análisis y resolución de problemas a través de computadoras.

**¿Como se trabaja?**

* Poseer un problema.
* Modelizar el problema.
* Modularizar la solución.
* Realizar el programa:

**Programa = Algoritmo + datos.**

Un algoritmo es un conjunto de instrucciones a realizar sobre un autómata para alcanzar un resultado deseado en un tiempo finito.

Datos: Son valores de información que los necesitamos y a veces los transformamos para efectuar la correcta función del programa.

**Para el desarrollador:**

1. Operatividad: El programa debe realizar la función que se desea.
2. Legibilidad: El código debe ser fácil de leer y entender.
3. Organización: El código debe estar modularizado para una mejor distribución de subfunciones.
4. Documentación.

**Para la computadora:**

1. Debe tener instrucciones válidas.
2. Deben terminar.
3. No debe usar recursos inexistentes.

* Utilizar el programa.

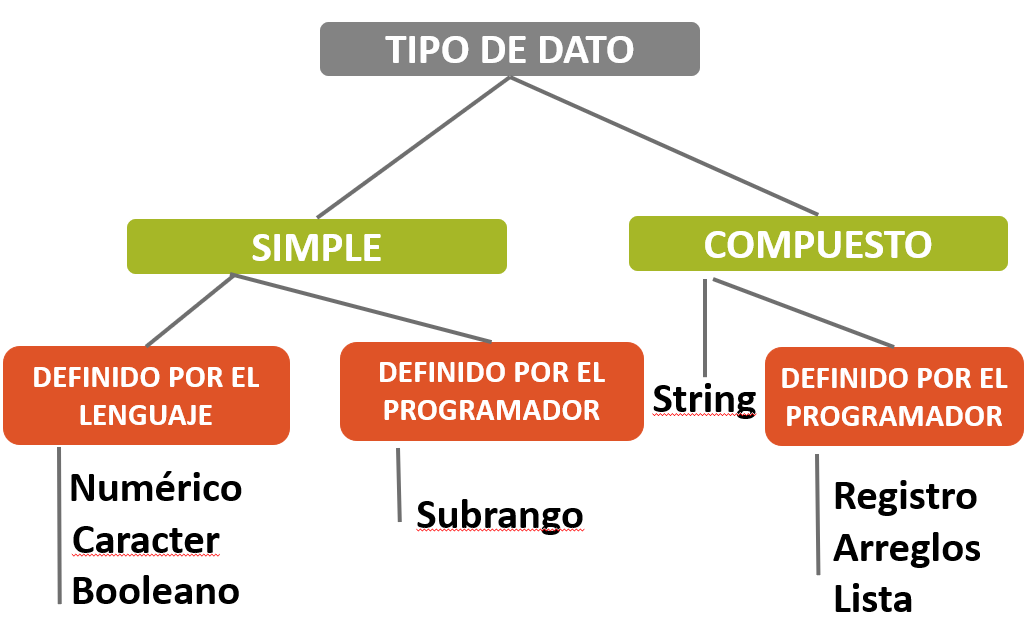
**Tipos de datos:** Es una clase de objetos de datos ligados a un conjunto de operaciones para crearlos y manipularlos.

**Simple:** Toman un único valor.

**Compuesto:** Pueden tomar varios valores a la vez.

**Definidos por el lenguaje:** Son provistos por el lenguaje.

**Definidos por el programador:** Permiten definir nuevos tipos de datos a partir de los simples.



**Tipo numérico:** Se pueden representar números enteros o reales.

Entero: Es simple y ordinal.

Operaciones para entero: matemáticas, lógicas, DIV y MOD.

Real: Es simple, permite representar decimales.

Operaciones para real: Matemáticas y lógicas.

**Tipo Lógico:** Permite representar datos que solo pueden tomar 2 valores, verdadero o falso. Es simple y ordinal. Permite operaciones lógicas (or, and, not).

**Tipo carácter:** Permite representar un carácter. Es simple y ordinal. Permite operaciones relacionales (=, <=, …).

**Tipos de variables:**

**Variable:** Es una zona de memoria que contiene alguno de los tipos anteriores. Puede cambiar su valor durante el programa.

**Constante:** Es una zona de memoria que contiene alguno de los tipos anteriores. No puede cambiar su valor durante el programa.

Los diferentes tipos de datos deben declararse siempre.

Una vez declarados debemos asociar al mismo variables.

**Pre condiciones:** Es la información que se conoce como verdadera antes de iniciar el programa.

**Post condiciones:** Es la información que debería ser verdadera al concluir el programa, si se cumplen adecuadamente los pasos.

**Estructuras de control**

**Read:** Se usa para tomar datos desde un dispositivo de entrada y asignarlos a una variable.

**Write:** Se usa para mostrar el contenido de una variable en pantalla.

**Estructura de control:** Es un conjunto de instrucciones que permiten especificar el control del algoritmo.

Existen: Secuencia, decisión, iteración, repetición y selección.

1. **Secuencia:** Está representada por una sucesión de operaciones.
2. **Decisión:** Es una estructura que se utiliza para la toma de decisiones, donde si la condición es verdadera se realizan ciertas acciones y si es falsa puede que se realicen otras. (If).
3. **Iteración:** Se utiliza para cuando no se sabe cuantas veces se quieren ejecutar ciertas instrucciones.

Se dividen en:

* **Iteración pre – condicionales:** Evalúa la condición y si es verdadera se ejecutan las acciones. (While).
* **Iteración post – condicionales:** Ejecuta las acciones, luego evalúa la condición y ejecuta las acciones si la condición es falsa. (Repeat).

1. **Repetición:** Ejecuta acciones un número de veces que ya se sabe de antemano. (For).
2. **Selección:** Permite realizar distintas acciones dependiendo del valor de una variable de tipo ordinal. (Case).

**Tipos de datos definidos por el Usuario (TDDU)**

**Subrango:** Es una sucesión de valores de un tipo ordinal. Es simple y ordinal.

Operaciones: Asignación, comparación y todas las permitidas para el tipo base.

**String:** Es una sucesión de caracteres. Es compuesto y ocupa una cantidad fija de memoria.

Operaciones: Asignación, comparación y operadores relacionales.

**Modularización**

Significa dividir un problema en subproblemas, para poder tener una mejor organización del programa y poder reutilizar módulos.

Cada subproblema puede resolverse independientemente y sus soluciones pueden combinarse para resolver el problema original.

**Módulo:** Tarea específica bien definida, se comunican entre sí y cooperan para conseguir un objetivo en común. Encapsula acciones, tareas o funciones.

Ventajas de la modularización: Mayor productividad, reusabilidad, facilidad de mantenimiento, facilidad de crecimiento y legibilidad.

**Procedures:** Conjunto de instrucciones que realizan una tarea específica y retorna 0, 1 o más valores.

**Function:** Conjunto de instrucciones que realizan una tarea específica y retorna un único valor de tipo simple.

**Alcance de variables:**

**Variables globales:** Pueden ser usada en todo el programa.

**Variables locales:** Pueden ser usadas solo en el proceso que están declaradas.

**Variables del programa:** Pueden ser usada solo en el cuerpo del programa principal.

**Modularización con parámetros**

Tipos de parámetros: Por valor y por referencia.

**Parámetro por valor:** Es llamado IN y significa que el módulo recibe un valor proveniente de otro módulo o del programa principal. El valor solo se modifica dentro del módulo, no hay ningún cambio en el valor que se recibe fuera del módulo.

**Parámetro por referencia:** El módulo recibe la dirección de una variable conocida en otros módulos. Si es modificado su valor, se verá reflejado en los demás módulos que conocen a la variable.

Características: El número y tipo de los argumentos utilizados en la invocación a un parámetro deben coincidir con el número y tipo de parámetros del encabezamiento del módulo.

**Registros**

Es uno de los tipos de datos estructurados, permite agrupar diferentes clases de datos en una estructura única.

Es heterogénea, los elementos pueden ser de distintos tipos.

Es estática, el tamaño no cambia durante la ejecución.

Tiene campos que representan cada uno de los datos que forman el registro.

**Arreglos**

Es una estructura de datos compuesta que permite acceder a cada componente a través de una variable índice, el cual da la posición de dicho componente.

Es homogénea: Los elementos son de un solo tipo.

Es estática: El tamaño no cambia durante la ejecución.

Es indexada: Para acceder a un elemento del arreglo es necesaria una variable índice para conocer su posición.

2 tipos de recorridos:

**Recorrido total:** Implica analizar todos los elementos del vector, o sea recorrerlo todo.

**Recorrido parcial:** Implica analizar los elementos del vector, hasta encontrar aquel que cumple con lo pedido. Puede que se recorra todo el vector.

2 dimensiones:

**Dimensión Física:** Se especifica en el momento de la declaración y determina su ocupación máxima de memoria.

**Dimensión Lógica:** Se determina cuando se cargan contenidos a los elementos del arreglo. No supera la dimensión física.

**Pasos para agregar:**

1. Verificar si hay espacio.
2. Agregar al final del elemento.
3. Incrementar la cantidad de elementos.

**Pasos para insertar:**

1. Verificar si hay espacio.
2. Verificar que la posición sea válida.
3. Insertar el elemento en la posición, tengo que hacer lugar.
4. Incrementar la cantidad de elementos.

**Pasos para borrar:**

1. Verificar que la posición sea válida.
2. Realizar el corrimiento.
3. Decrementar la cantidad de elementos.

Búsqueda:

**Vector desordenado:** Se debe recorrer el vector hasta encontrar el dato o hasta que se termine el vector.

**Vector ordenado:** Se aprovecha el orden. 2 formas: Búsqueda mejorada y búsqueda dicotómica.

**Punteros**

**Variables estáticas:** No permite que las estructuras varíen su dimensión. Las variables y tipos reservan memoria en su declaración y se mantienen durante todo el programa.

**Variables dinámicas:** Permite modificar en ejecución la memoria utilizada.

**Puntero:** Una variable usada para almacenar una dirección de memoria de otra variable. Es de tipo simple.

**Variable puntero:** Ocupa una cantidad fija de memoria, independiente del tipo de dato al que apunta. Puede reservar y liberar memoria durante la ejecución del programa.

**Variable referenciada:** No tienen memoria asignada.

Dispose (p): Libera la conexión entre la variable y la posición de memoria. También libera la posición de memoria.

P:= nil: Libera la conexión entre la variable y la posición de memoria. Pero la memoria sigue ocupada.

Operaciones: (p=nil), (p=q), (p^=q^).

**Lista**

Colección de elementos homogéneos, vinculados con una relación lineal, cada elemento tiene un único predecesor (Excepto el primero) y un único sucesor (Excepto el último).

Está compuestas por nodos, que se conectan mediante punteros. Se agregan nodos con “new”. Cuando hay nodos que ya no se necesitan pueden ser borrados, liberando memoria con dispose.

|  |  |
| --- | --- |
| **Arreglo** | **Lista** |
| Estructura estática | Estructura dinámica |
| Homogénea | Homogénea |
| Indexada, de acceso directo. | Lineal, acceso no directo. |
| Agregar adelante: Debe hacerse un corrimiento para generar el espacio y aumentar la dimensión lógica. Puede que no se pueda hacer. | Agregar adelante: Se debe solicitar un espacio y reacomodar los punteros. Siempre se puede hacer. |
| Agregar al final: Se agrega el elemento en la dimensión lógica + 1 y se incrementa la dimensión lógica. Puede que no se pueda hacer. | Agregar al final: Recorrer toda la lista, solicitar espacio y reacomodar los punteros; O también se puede llevar un puntero con el último para no tener que recorrer toda la lista. Siempre se puede hacer. |
| Insertar: Buscar la posición, luego correr para hacer lugar, se carga el elemento y se incrementa la dimensión lógica. Puede que no se pueda hacer. | Insertar: Se solicita espacio, se busca la posición, una vez encontrada se reacomodan los punteros. Siempre se puede hacer. |
| Eliminar: Se busca el elemento, se realiza un corrimiento desde la posición donde está el elemento hasta el final y se decrementa la dimensión lógica. | Eliminar: Se busca el elemento, se elimina y se reorganizan los punteros. |
| Memoria ocupada: Siempre ocupa la misma cantidad de memoria estática. | Memoria ocupada: Su tamaño cambia al agregar o eliminar elementos. |
| Parámetros en las operaciones: Cuando un arreglo es pasado por valor, se copia todo el arreglo.  Cuando es pasado por referencia se pasa solo la dirección. | Parámetros en las operaciones: Cuando una lista es pasada por valor se copia la dirección inicial de la lista.  Cuando es pasada por referencia se pasa la dirección inicial de la lista. |

**Eficiencia y corrección**

**Corrección de programas:** Un programa es correcto si se realiza de acuerdo a sus especificaciones.

Técnicas para verificar si un programa es correcto:

1. Testing: Es proveer evidencias convincentes de que el programa hace el trabajo esperado.
2. Debugging: Es el proceso de descubrir y reparar la causa del error.
3. Walkthroughs: Es recorrer el programa frente a una audiencia.
4. Verificación: Controlar que se cumplan las pre y post condiciones del programa.

**Eficiencia de programas:** Se estudia el tiempo que tarda un algoritmo en ejecutarse y la memoria que requiere.

La eficiencia se relaciona con: datos de entrada (tamaño y cantidad), calidad del código generado por el compilador, naturaleza y rapidez en la ejecución de las instrucciones de máquina y el tiempo del algoritmo base.

Existen dos formas para calcular el tiempo de un algoritmo:

**Análisis empírico:** Es necesario realizar el programa y medir los recursos consumidos.

Características: Fácil de realizar, obtiene valores exactos para una máquina y unos datos determinados, completamente independiente de la máquina donde se ejecuta y requiere implementar el algoritmo y ejecutarlo repetidas veces.

**Análisis teórico:** Se define T(n) al tiempo de ejecución de un programa con una entrada de tamaño n. Las unidades de T(n) no se especifican.

¿Cómo se calcula?

Regla 1: For. N\*Max(i1,i2)

Regla 2: For anidados.

Regla 3: Sentencias consecutivas. Max (inst1, inst2)

Regla 4: If/Else. Max (if, else)