**Definiciones generales**

**Informática**: Ciencia que estudia el análisis y resolución de problemas utilizando computadoras.

**Computadora**: Máquina digital y sincrónica con capacidad de cálculo numérico y lógico controlado por un programa almacenado y con comunicación con el entorno.

**Programa:** Conjunto de instrucciones expresadas en un lenguaje informático capaz de ser interpretado por una computadora para cumplir una tarea específica.

**Pre condición**: Información que se establece como verdadera antes de iniciar un programa o módulo.

**Post** **condición**: Información que debería ser verdadera al concluir el programa o módulo si éste cumplió con el propósito esperado.

**Algoritmo**: Conjunto de acciones que debe implementar la computadora para cumplir con un propósito específico.

**Dato**: Representación de un objeto del mundo real que permite modelizar aspectos del problema a resolver con un programa sobre una computadora. Puede ser constante o variable, simple o compuesto, definido por el lenguaje o definido por el programador.

**Variable**: Zona de memoria que almacena un dato que puede cambiar su valor durante el programa.

**Constante**: Zona de memoria que almacena un dato que no cambia de valor durante el programa

**Modularización**: Descomposición de un problema en bloques de instrucciones funcionales e independientes menos complejas, más legibles y reutilizables.

**Módulo**: Bloque de instrucciones funcionalmente independientes que facilita la resolución de un algoritmo, mejora la lectura de un programa y puede ser reutilizada.

**Procedimiento**: Módulo de instrucciones funcionalmente independiente que realiza una tarea específica y retorna 0, 1 o más valores.

**Función**: Módulo de instrucciones funcionalmente independiente que realizan una tarea específica y retornan un único valor de tipo simple.

**Parámetro**: Modo por el cual los módulos pueden compartir información con otros módulos y con el programa principal.

**Estructuras** **de** **control**: Instrucciones que controlan el orden de ejecución del algoritmo a implementar por un programa.

**Estructura** **de** **datos**: Conjunto de datos que tienen valores lógicamente relacionados y asociados bajo un nombre único. Puede ser de elementos homogéneos o heterogéneos, de acceso directo o secuencial, de tamaño estático o dinámico; lineal o no lineal.

**Registro**: Es una estructura de datos compuesta definida por el usuario, estática (su extensión no varía) y heterogénea (permite agrupar diferentes clases de datos bajo un sólo nombre) que solo puede leerse campo por campo.

**Arreglo**: Estructura de datos homogénea (con datos del mismo tipo), estática (su dimensión física no varía), e indexada (el acceso a sus componentes es a través de una variable índice que da su posición dentro de ella).

**Dimensión** **física**: Cantidad total de memoria reservada para un arreglo, la cual se establece de antemano y permanece invariable durante la ejecución del programa.

**Dimensión** **lógica**. Cantidad de posiciones de memoria ocupadas por el contenido real de un arreglo. Puede variar durante la ejecución de un programa pero no superar la dimensión física.

**Puntero**: Tipo de dato simple que contiene la dirección de memoria donde se halla un valor que puede ser de cualquier tipo disponible (char, boolean, integer, real, string, registro, arreglo, lista u otro puntero).

**Lista**: Tipo de dato compuesto definido por el programador que consiste en un conjunto de nodos, donde cada nodo contiene un elemento y la dirección de memoria donde se encuentra el siguiente nodo. Los nodos que componen una lista pueden no ocupar posiciones contiguas de memoria pero mantienen un orden lógico interno.

**Corrección**: Condición que cumple un programa cuando realiza la función para la cual fue desarrollado.

**Eficiencia**: Capacidad de un programa de hacer un uso eficiente de la memoria y optimizar el tiempo de ejecución.

**Datos**

Características

* Tienen un rango de valores posibles.
* Tienen un conjunto de operaciones permitidas.
* Tienen una representación interna.

**Tipos de datos**

**Simples**: Aquellos que toman un único valor, en un momento determinado, de todos los permitidos para ese tipo. Ejemplo: real, integer.

**Compuestos**: Pueden tomar varios valores a la vez que guardan alguna relación lógica entre sí bajo un único nombre. Ejemplo: string, registros, arreglos.

**Definidos** **por** **el** **lenguaje**: Son provistos por el lenguaje y tanto la representación como sus operaciones y valores son reservadas al mismo. Ejemplo: integer, char, string.

**Definidos por el programador**: Es un tipo de dato que no existe en la definición del lenguaje y que le permite al programador crear datos no estándar a partir de los tipos simples. Ejemplo: subrango, registros, arreglos

**Numéricos**: Representan el conjunto de números y pueden ser enteros o reales. Con ellos es posible realizar operaciones matemáticas (+ - \* /), lógicas (> < =), y con operadores enteros (MOD, DIV). Los enteros, son simples y ordinales; los reales, son sólo simples

**Lógicos**: Tipo de dato simple que puede tomar sólo dos valores: verdadero o falso. Con él se pueden realizar operaciones lógicas (AND, OR, NOT).

**Caracter**: Representa un conjunto finito y ordenado de caracteres que la computadora reconoce. Es de tipo simple y ordinal. Con él se pueden realizar operaciones lógicas (< > =).

**Subrango**: Tipo de dato compuesto y definido por el usuario que consiste de una sucesión ordenada de valores (por lo que es ordinal) y de un mismo tipo (homogéneo) tomado como base. Las operaciones que se permiten con ellos son la asignación, la comparación y aquellas permitidas para su tipo base.

**Estructuras de control**

Son el conjunto de instrucciones que controlan el orden de ejecución del algoritmo a implementar.

**Secuencia**: Estructura de control representada por una sucesión de operaciones en la que el orden de ejecución coincide con el orden de aparición.

**Decisión**: Estructura que obliga a optar entre dos alternativas, cada una de las cuales conduce a otra instrucción o serie de instrucciones distintas (if).

**Selección**. Estructura que permite realizar distintas acciones dependiendo del valor de una variable de tipo ordinal (case).

**Iteración**: Estructura que control que, en función de la evaluación de una condición previa o posterior, ejecuta o no un bloque de acciones determinada cantidad de veces. Cuando es precondicional la iteración se ejecuta mientras la condición es falsa (while); si es postcondicional se ejecuta hasta que la condición se vuelve verdadera (repeat until).

**Repetición**: Es la estructura que permite repetir N veces una acción o bloque de acciones. El número de veces es fijo y conocido de antemano (for).

**Modularización**

Ventajas

* Mayor productividad
* Reusabilidad
* Facilidad de mantenimiento
* Legibilidad
* Facilidad de escalamiento

**Tipos de módulos**

* **Función**: Módulo de instrucciones funcionalmente independiente que realiza una tarea específica y retorna un único valor de tipo simple.
* El resultado se asigna a una variable del mismo tipo que devuelve la función
* El retorno de la función es a la misma línea de invocación
* **Procedimiento**: Módulo de instrucciones funcionalmente independiente que realiza una tarea específica y retorna 0, 1 o más valores, lo que admite datos de tipo compuesto.

**Parámetros**

Es la forma en que los módulos comparten datos con otros módulos o con el programa principal.

**Por** **valor**: También llamado parámetro IN es un valor que el módulo recibe (sobre una variable local) desde otro módulo o desde el programa principal. Con él puede realizar operaciones y/o cálculos pero no producirá ningún cambio ni tampoco tendrá incidencia fuera del propio módulo, ya que funciona por copia de un valor.

**Por** **referencia**: También llamado INOUT es una variable que le es compartida al módulo desde otro módulo o desde el programa principal. Puede operar con ella y las modificaciones que se produzcan se reflejan en los demás módulos que conocen la variable ya que funciona por referencia a una dirección que almacena esa variable.

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

**Alcance de las variables**

**Variables globales**: Pueden ser usadas en todo el programa, incluyendo módulos.

**Variables locales al programa**: Pueden ser usadas sólo en el cuerpo del programa.

**Variables** **locales** **al** **proceso**: Pueden ser usadas sólo en el proceso que las declaró.

Si es una variable utilizada en un proceso

* Se busca si es variable local
* Se busca si es un parámetro
* Se busca si es variable global

Si es una variable utilizada en un programa

* Se busca si es variable local
* Se busca si es un parámetro
* Se busca si es variable global

**Punteros**

Son un tipo de dato simple que contiene la dirección de memoria donde se halla un valor que puede ser de cualquiera de los tipos vistos (char, boolean, integer, real, string, registro, arreglo u otro puntero ).

Ocupan una cantidad de memoria fija de 4 bytes independiente al tipo de dato al que apuntan porque sólo contienen una dirección de memoria en hexadecimal. Pueden reservar y liberar memoria durante la ejecución del programa.

**Operaciones posibles con punteros**

* Crear un puntero implica reservar una dirección de memoria dinámica libre para almacenar en ella los contenidos asociados a esa variable puntero. (new (p))
* Eliminar un puntero implica liberar la memoria dinámica que contenía la variable de tipo puntero. (dispose (p))
* Liberar un puntero implica cortar el enlace que tiene con la memoria dinámica. Esta queda ocupada pero ya no se puede acceder. (p:= nil)
* Asignar un puntero implica asignar la dirección de un puntero a otra variable puntero del mismo tipo.
* Acceder al contenido de un puntero implica operar sobre el contenido de la dirección de memoria a la que apunta el puntero. (p^)

Operaciones que no se pueden hacer

* Leer un puntero (read)
* Imprimir el puntero (write)
* Asignarles una dirección en forma manual (p:= AF13)
* Comparar sus direcciones por mayor o menor (p<q)

**Lista**

Es un tipo de dato compuesto definido por el programador que consiste en un conjunto de nodos, donde cada nodo contiene un elemento y la dirección de memoria donde se encuentra el siguiente nodo. Cada nodo está representado por un puntero y toda lista tiene un nodo inicial. Los nodos que componen una lista pueden no ocupar posiciones contiguas de memoria pero mantienen un orden lógico interno.

Características

* Homogénea: Todos sus elementos son del mismo tipo
* Dinámica: La cantidad de elementos puede variar durante el programa.
* Lineal: Cada nodo tiene un único antecesor y sucesor.
* Acceso secuencial: Para llegar a un elemento hay que pasar por los anteriores.

Operaciones posibles

* Crear una lista.
* Agregar nodos al comienzo de la lista.
* Recorrer una lista.
* Agregar nodos al final de la lista.
* Insertar nodos en una lista ordenada
* Eliminar nodos de una lista

**Cálculo de memoria**

La memoria de un programa es la cantidad de memoria necesaria para que pueda ejecutarse y se divide en dos.

**Memoria estática**: es la requerida por las variables locales y globales del programa

**Memoria dinámica**: es requerida por instrucciones del programa que reservan o liberan memoria.

**Variables estáticas**: Son aquellas que reservan memoria en su declaración y no la modifican mientras se ejecuta el programa.

**Variables dinámicas**: Son aquellas que modifican la memoria reservada para ellas durante la ejecución del programa.

**Tabla de ocupación**:

Char (1 byte)

Boolean, (1 byte)

Integer (4 bytes)

Real (8 bytes)

String, (tamaño + 1 byte)

Subrango, (depende el tipo)

Registro, (suma de sus campos)

Arreglos (dimFísica \* tipo elemento)

Puntero (4 bytes)

**Corrección y Eficiencia**

Un programa es correcto cuando cumple con la función para la cual fue desarrollado

Un programa es eficiente optimiza el tiempo de ejecución y el uso de la memoria

**Técnicas de revisar la corrección de programas**

* **Testing**: Consiste en diseñar e implementar un conjunto de pruebas para comprobar que los distintos aspectos de un programa funcionen como se espera poniendo atención a los casos límites.
* **Debbugging**: Consiste en agregar al código del programa sentencias de impresión para monitorear qué está pasando en distintos momentos de él
* **Walkthroughs**: Consiste en recorrer un programa frente a una audiencia para ir analizando sus aspectos entre todos.
* **Verificación**: Consiste en verificar que el programa cumple con las pre y post condiciones establecidas.

**Tiempo de ejecución**

Cómo se calcula

* Sólo se consideran las instrucciones elementales del algoritmo: asignación, operaciones aritméticas y operaciones lógicas.
* Cada instrucción utiliza un tiempo constante equivalente a una unidad de tiempo (UT).
* Cuando un algoritmo tiene un tiempo de ejecución constante por no incluir estructuras de control iterativas sólo se suman las unidades de tiempo de sus sucesivas instrucciones.
* Con el IF, el tiempo de ejecución se determina considerando el peor de los casos y se calcula una unidad de tiempo por cada condición a evaluar. La fórmula es: condiciones + MAX (if/else).
* Con el FOR se aplica un cálculo fijo que consiste en multiplicar la cantidad de ciclos por 3 y sumarle 2. Y a eso sumarle el valor del cuerpo multiplicado por la cantidad de ciclos. La fórmula es: 3 (N)+2 + N (cuerpo). Cuando no sabemos cuánto vale N, el resultado se expresa incluyendo a N como un valor incógnito.
* Con el WHILE también se aplica una fórmula fija que es: N+1 (condición) + N (cuerpo).

**Algoritmos**

**Agregar en un vector**

• Verificar que la dimensión lógica sea menor a la física para ver si hay espacio.

• Agregar el elemento nuevo al final de los elementos ya existentes.

• Incrementar la dimensión lógica para que contemple el elemento agregado.

**Insertar en un vector**

• Verificar que la dimensión lógica sea menor a la física para ver si hay espacio.

• Verificar que la posición indicada para insertar sea válida constatando que esté entre los valores abarcados por la dimensión lógica.

• Desplazar una posición hacia atrás los elementos subsiguientes a la posición indicada para hacer lugar al nuevo elemento.

• Insertar el nuevo valor en la posición indicada.

• Incrementar la dimensión lógica para que contemple el elemento agregado.

**Eliminar en un vector**

• Verificar que la posición indicada para eliminar sea válida constatando que esté entre los valores abarcados por la dimensión lógica.

• Desplazar una posición hacia adelante todos los elementos existentes desde la posición siguiente a la que hay que eliminar hasta la dimensión lógica.

• Decrementar la dimensión lógica para que refleje el elemento eliminado.

**Agregar adelante en una lista**

• Crear un nuevo nodo.

• Cargar el nuevo nodo con el dato que corresponda y la dirección siguiente ‘nil’

• Verificar si el nodo a agregar va a ser el primero de la lista o ya existen otros

• Si va a ser el primero, asignar al puntero inicial la dirección del nuevo nodo.

• Si la lista ya tenía otros nodos, indicar que el siguiente del nuevo nodo es el puntero inicial y actualizar el puntero inicial con la dirección del nuevo nodo.

**Agregar atrás en una lista**

• Crear un nuevo nodo.

• Cargar el nuevo nodo con el dato que corresponda y la dirección siguiente ‘nil’

• Verificar si el nodo a agregar va a ser el primero de la lista o ya existen otros.

• Si va a ser el primero, asignar al puntero inicial la dirección del nuevo nodo.

• Si no, inicializar un puntero auxiliar para recorrer la lista hasta ubicarse en su último nodo.

• Actualizar como siguiente del último nodo al nuevo elemento

**Buscar un elemento en una lista**

• Inicializar una variable auxiliar con la dirección del puntero inicial de la lista

• Inicializar una variable booleana en falso

• Recorrer la lista nodo por nodo mientras no sea el final y no encuentre el elemento buscado

• Indicar que en caso de dar con el elemento buscado la variable booleana pase a ser verdadera

**Insertar un elemento en una lista**

• Se necesita que la estructura tenga un orden e implica agregar el elemento a la lista de manera que la misma siga ordenada.

• Generar un nuevo nodo.

• Cargar el nuevo nodo con el dato que corresponda y la dirección siguiente ‘nil’

• Verificar que la lista no esté vacía

• Si la lista está vacía, asigna al nuevo nodo la dirección del puntero inicial.

• Si la lista no está vacía, buscar la posición a insertar recorriéndola con dos punteros auxiliares que indiquen el nodo actual y el anterior.

• Hacer que campo siguiente del puntero anterior apunte a la dirección del nuevo nodo, y que el campo siguiente del nuevo nodo apunte a la dirección del puntero actual.

**Eliminar un elemento de una lista**

• Inicializar dos punteros auxiliares (anterior y actual) con la dirección del puntero inicial de la lista

• Recorrer con ellos la lista nodo por nodo mientras no se llegue a final y no se encuentre el elemento a eliminar.

• Si el elemento a eliminar resultara ser el primero de la lista, se actualiza el puntero inicial y se hace un dispose del nodo a eliminar.

• Si el elemento a eliminar estuviera en el medio de la lista, el campo siguiente del puntero auxiliar anterior pasa a tener la dirección del campo siguiente del puntero auxiliar actual y se elimina con un dispose el nodo actual.