UNIVERSIDAD TECNOLOGICA NACIONAL | FACULTAD REGIONAL SAN FRANCISCO

Tecnicatura Universitaria en Programación

Programación 1

Plan 2024

Eje temático Nº 3

Unidad Nº 6: Programación Imperativa: Estructuras de Datos.

Vectores

Son estructuras de datos fundamentales en programación que nos permiten almacenar una colección ordenada de elementos del mismo tipo. También se les conoce como arreglos.

Estas son algunas características importantes de los vectores:

- **Colección ordenada:** Los elementos en un vector están dispuestos en un orden específico. Cada elemento tiene una posición única en el vector, que se conoce como índice.
- **Tipo de datos de elementos:** En un lenguaje fuertemente tipado (por ej.: C#), todos los elementos en un vector deben ser del mismo tipo de datos. Por ejemplo, un vector de enteros solo puede contener números enteros, un vector de caracteres sólo puede contener caracteres, y así sucesivamente. En cambio en un lenguaje débilmente tipado (por ej.: JavaScript), los elementos del vector pueden ser de cualquier tipo.
- **Acceso mediante índices:** Podemos acceder a los elementos individuales de un vector utilizando su índice. El primer elemento tiene el índice 0, el segundo tiene el índice 1, y así sucesivamente. El índice simplemente es el número de la posición del elemento dentro del vector.
- **Tamaño fijo:** En muchos lenguajes de programación, el tamaño de un vector se define al crearlo y no se puede cambiar después. Sin embargo, algunos lenguajes de programación proporcionan estructuras de datos similares a los vectores con tamaños dinámicos, que pueden cambiar de tamaño durante la ejecución del programa.

Los vectores son útiles para almacenar y manipular conjuntos de datos relacionados, como una lista de nombres, una serie de valores numéricos, o los resultados de una encuesta. Son ampliamente utilizados en programación y forman la base para muchas otras estructuras de datos más complejas.

Por ejemplo, en un vector de 5 elementos, podemos ver que cada elemento tiene un índice único y secuencial, comenzando por cero.

Automóvil	Motocicleta	Avión	Barco	Helicóptero
0	1	2	3	4

Para recrear este vector en C#, necesitamos declararlo y definir su cantidad.

C#

```
1    string[] vehiculos = { “Automóvil”, “Motocicleta”,“Avión”,
     “Barco”,“Helicóptero” };
```

Esta forma de declarar el vector en una misma línea, nos ahorra tiempo para definirla y dejarla cargada en un solo paso. En vectores de pocos elementos, suele ser práctico. En cambio en vectores que requieren almacenar muchos elementos, no es la opción más recomendada.

Para consumir un elemento del vector, utilizamos los corchetes y la posición del índice. Si queremos imprimir en pantalla el primer elemento del vector, lo consultamos así:

C#

```
1    Console.WriteLine(vehiculos[0]);
```

Esta línea de C# imprime en consola el dato cargado en la posición cero del vector de vehiculos.

Consola

```
Automóvil
```

Visto de otra manera, el ejemplo de elementos para este nombre de variable, sería el siguiente:

Elemento >>	Automóvil	Motocicleta	Avión	Barco	Helicóptero
índice >>	0	1	2	3	4
variable >>	vehiculos[0]	vehiculos[1]	vehiculos[2]	vehiculos[3]	vehiculos[4]

Ejercicios

Ejercicio 1

Cargar números en un vector de 5 valores, luego mostrar la suma y el promedio.

Ejercicio 2

Cargar un vector de 15 números, sumarle a las posiciones pares 2 y a las impares 3 y reemplazar los valores del vector. Finalmente mostrar todos los valores del vector resultante.

Ejercicio 3

Cargar dos vectores VECTOR1 y VECTOR2 de 10 elementos cada uno con valores ingresados por el operador.
Analizar si son iguales (se consideran iguales cuando cada elemento de VECTOR1 es igual a cada elemento correspondiente de VECTOR2).
Mostrar finalmente ‘Los vectores SON iguales’ o ‘Los vectores NO son iguales’, de acuerdo al resultado.

Matrices

Las matrices son estructuras de datos bidimensionales que consisten en filas y columnas, organizadas en forma de tabla. Son una extensión natural de los vectores unidimensionales (o arreglos), ya que permiten almacenar datos de manera ordenada, pero en múltiples dimensiones.

Aquí hay algunas características importantes de las matrices:

Filas y columnas: Una matriz está compuesta por filas y columnas. Cada fila contiene un conjunto de elementos del mismo tipo, y cada columna representa una propiedad o dimensión de los datos.

Elementos del mismo tipo: Todos los elementos de una matriz deben ser del mismo tipo de datos, al igual que en los vectores.

Índices bidimensionales: Para acceder a un elemento específico en una matriz, se utilizan dos índices: uno para la fila y otro para la columna. El primer elemento se encuentra en la fila 0 y la columna 0, y así sucesivamente.

Tamaño definido: Al igual que los vectores, el tamaño de una matriz se define al crearla y no se puede cambiar después.

Las matrices son muy útiles para almacenar datos estructurados en forma de tabla, como una cuadrícula de números, una matriz de adyacencia en grafos, o una colección de datos en dos dimensiones. Se utilizan ampliamente en campos como la programación, las matemáticas, la ingeniería y la ciencia de datos para modelar y manipular información de manera eficiente.

Elemento >>	Automóvil 0,0	Motocicleta 0,1	Avión 0,2	Barco 0,3	Helicóptero 0,4
Elemento >>	Automóvil 1,0	Motocicleta 1,1	Avión 1,2	Barco 1,3	Helicóptero 1,4

string[,] nombreMatriz = new string[2, 3];

Ejercicios

Ejercicio 1

Escribir un programa que sume los elementos de cada fila de una matriz de 15 filas x 20 columnas y muestre los resultados por fila.

Ejercicio 2

Escribir un programa que muestre el promedio de los elementos de cada columna de una matriz de 20 filas x 10 columnas y muestre los resultados por columna.

Ejercicio 3

Escribir un programa que sume los elementos de la diagonal principal que va de izquierda a derecha en una matriz de m x n.

Unidad Nº 7: Programación Funcional, Abstracciones con procedimientos y funciones.

La resolución de problemas complejos se facilita considerablemente si se dividen en problemas más pequeños; y la resolución de estos subproblemas se realiza mediante subalgoritmos.

Los subalgoritmos son unidades de programa o módulos que están diseñados para ejecutar alguna tarea específica. Éstos, constituidos por funciones o procedimientos, se escriben solamente una vez, pero pueden ser referenciados en diferentes puntos del programa, de modo que se puede evitar la duplicación innecesaria del código.

El módulo principal se ejecuta en una primera instancia, que da la orden de inicio de ejecución de los subprogramas. Puede ser ejecutado n veces. Es importante saber que datos se van a compartir entre los programas.

El subprograma es un programa en sí mismo, ejecutado por la solicitud del programa principal o de otro subprograma, una n cantidad de veces. Cuando realiza la solicitud, el programa se detiene hasta que el subprograma deja de realizar su tarea, luego continúa; esto se conoce como control de ejecución.

FUNCIONES

Una función es un subprograma que recibe, como argumentos o parámetros, datos de tipo numérico o no numérico, y devuelve un único resultado.

Las funciones incorporadas al sistema se denominan funciones internas, o intrínsecas; las funciones definidas por el usuario se llaman funciones externas.

El algoritmo o programa invoca la función con el nombre de esta última en una expresión seguida de una lista de argumentos que deben coincidir en cantidad, tipo y orden con los de la función que fue definida.

PROCEDIMIENTOS

Un procedimiento es un subprograma que ejecuta una tarea determinada. Está compuesto por un conjunto de sentencias, a las que se le asigna un nombre, o identificador. Constituyen unidades del programa, y su tarea se ejecuta siempre y cuando encuentre el nombre que se le asignó a dicho procedimiento.

Los procedimientos deben ser declarados obligatoriamente antes de que puedan ser llamados en el cuerpo del programa principal. Para ser activados o ejecutados, deben ser llamados desde el programa en que fueron declarados.

Pueden recibir cero o más valores del programa principal que lo llama y lo activa.

Las ventajas más destacables de usar procedimientos son:

- Se pueden ejecutar más de una vez en un programa, con solo llamarlos las veces que así desee. Con esto se ahorra tiempo de programación.
- El mismo procedimiento se puede usar en distintos programas.
- Su uso facilita la división de tareas entre los programadores de un equipo.
- Se pueden probar individualmente e incorporarlos en librerías o bibliotecas.

Ejercicios

Ejercicio 1

Realizar un método C# para mostrar un menú de opciones de un sistema

Ejercicio 2

Realizar un método C# que devuelva un texto ingresado por el usuario, el mismo NO debe estar vacío, ni ser espacios en blanco

Ejercicio 3

Realizar un método C# que devuelva una opción de un menú (ejercicio 1) en donde el usuario pueda ingresar una opción, de no ser válida volver a solicitarla

Unidad Nº 8: Programación Funcional: Recursividad.

Un subprograma que se puede llamar a sí mismo se llama recursivo. La recursión puede ser utilizada como una alternativa a la repetición o estructura repetitiva. La escritura de un procedimiento o función recursiva es similar a sus homónimos no recursivos; sin embargo, para evitar que la recursión continúe indefinidamente, es preciso incluir una condición de terminación.

1- Define un caso base: Esto establece la condición en la que la función recursiva deja de llamarse a sí misma y devuelve un resultado directo.
2- Divide el problema en subproblemas más pequeños: Esto implica descomponer el problema original en instancias más simples del mismo problema o en problemas relacionados.
3- Llama a la función recursivamente: Utiliza la función dentro de sí misma para resolver los subproblemas generados en el paso anterior.
4- Combina los resultados de los subproblemas: Una vez que se han resuelto los subproblemas, se combinan para obtener el resultado final.

La programación recursiva puede ser poderosa, pero debe manejarse con cuidado para evitar el riesgo de bucles infinitos o un consumo excesivo de recursos.

Ejercicios

Ejercicio 1

Calcular la potencia de un número x elevado a n (es decir, x^n) de manera recursiva.

Ejercicio 2

Sumar los elementos de un array de enteros de manera recursiva.

Ejercicio 3

Invertir una cadena de texto utilizando recursión.

Unidad Nº 9: Programación Imperativa: Algoritmos de Búsqueda, Recorrido y Ordenamiento.

Algoritmos de búsqueda:

Es aquel que está diseñado para localizar un elemento concreto dentro de una estructura de datos.
Consiste en solucionar un problema booleano de existencia o no de un elemento en un determinado conjunto finito de elementos.
Además en caso de existir, el algoritmo podría proporcionar la localización del elemento dentro del conjunto.

Búsqueda lineal o secuencial: Recorre secuencialmente cada elemento de una lista hasta encontrar el valor buscado o llegar al final de la lista.

Búsqueda binaria: Requiere que la lista esté ordenada. Divide repetidamente la lista por la mitad y compara el valor buscado con el elemento medio, reduciendo así la búsqueda a una mitad de la lista.

Ejercicios

Ejercicio 1

Escribir un algoritmo que utilizando búsqueda secuencial informe la cantidad de ocurrencias del dato x (ingresado por el operador) en un vector de 500 elementos.

Ejercicio 2

Se tienen 3 vectores con legajos, promedios de notas, cantidad de materias aprobadas
Escribir un algoritmo que lea un legajo, lo busque y si lo encuentra informe el promedio y cantidad de materias aprobadas de ese alumno. Caso contrario mostrar un mensaje indicando dicha situación.

Algoritmos de ordenamiento:

Es la operación de organizar un conjunto de datos en algún orden dado, tal como creciente o decreciente en datos numéricos o bien en orden alfabético.

Burbuja (Bubble Sort): Compara pares de elementos adyacentes y los intercambia si están en el orden incorrecto, repitiendo este proceso hasta que la lista esté ordenada.

Selección (Selection Sort): Encuentra el elemento más pequeño y lo intercambia con el primer elemento. Luego, encuentra el segundo elemento más pequeño y lo intercambia con el segundo lugar, y así sucesivamente.

Inserción (Insertion Sort): Construye una lista ordenada uno a uno, tomando cada elemento de la lista sin ordenar e insertándolo en su lugar correcto en la lista ordenada.

Merge Sort: Divide la lista en mitades iguales, ordena cada mitad recursivamente y luego combina las mitades ordenadas en una sola lista ordenada.

Quick Sort: Elige un elemento llamado pivote y particiona la lista alrededor del pivote, de manera que los elementos más pequeños estén a un lado del pivote y los elementos más grandes estén al otro lado. Luego, ordena recursivamente las sublistas generadas.

Cada algoritmo tiene sus propias ventajas y desventajas en términos de eficiencia, uso de memoria y facilidad de implementación, lo que hace que ciertos algoritmos sean más adecuados para diferentes situaciones.

Ejercicios

Ejercicio 1

Definir un vector de números enteros y ordenarlo con el método burbuja

Ejercicio 2

Definir un vector de números enteros y ordenarlo con el método quick sort