Variable, Data types, Keywords, Array

Lesson Objectives

- Variable
- Keywords
- Data types
- Array

Before start

- Understand .NET Framework
- Visual Studio readiness
- Basic I/O

Section 1

VARIABLE

Variable

- A variable is used to store data in a program and is declared with an associated data type.
- A variable has a name and may contain a value.
- A data type defines the type of data that can be stored in a variable.

```
var age = 10;
Console.WriteLine(age);
```

Variable

- A variable is an entity whose value can keep changing during the course of a program.
  - ✓ Example: The age of a student, the address of a faculty member, and the salary of an employee are all examples of entities that can be represented by variables.
- In C#, a variable is a location in the computer's memory that is identified by a unique name and is used to store a value. The name of the variable is used to access and read the value stored in it.

Data Type

- Different types of values such as numbers, characters, or strings can be stored in different variables.
- To identify the type of data that can be stored in a variable, C# provides different data types.
- When a variable is declared, a data type is assigned to the variable.
- This allows the variable to store values of the assigned data type

Data Type

Value Type

- Stores the value in its memory space
- Primitive data types and struct
- Passes by value (default)

Reference Type

- Stores the address of the value where it is stored
- Class objects, string, array, delegates
- Passes by ref (default)

Null pointer

- Reference types have null value by default, when they are not initialized.
  - ✓ For example, a string variable (or any other variable of reference type datatype) without a value assigned to it.
  - ✓ In this case, it has a null value, meaning it doesn't point to any other memory location, because it has no value yet.

Null pointer

- Reference types have null value by default, when they are not initialized.
- A value type variable cannot be null because it holds a value not a memory address
  - ✓ Value type variables must be assigned some value before use
  - ✓ The compiler will give an error if you try to use a local value type variable without assigning a value to it

Boxing and Unboxing

- Boxing is the process of converting a value type to the type object or to any interface type implemented by this value type.
  - ✓ When the common language runtime (CLR) boxes a value type, it wraps the value inside a System.Object instance and stores it on the managed heap.
- Unboxing extracts the value type from the object.
  - ✓ Boxing is implicit; unboxing is explicit.
- The concept of boxing and unboxing underlies the C# unified view of the type system in which a value of any type can be treated as an object.

Boxing and Unboxing

- Unboxing is an **explicit conversion** from the type object to a value type or from an interface type to a value type that implements the interface.
- An unboxing operation consists of:
  - ✓ Checking the object instance to make sure that it is a boxed value of the given value type.
  - ✓ Copying the value from the instance into the value-type variable.

Pre-defined Data Types

- The pre-defined data types are referred to as basic data types in C# that have a pre-defined range and size.
- The size of the data type helps the compiler to allocate memory space and the range helps the compiler to ensure that the value assigned, is within the range of the variable's data type.

Value Types

Data Type	Size	Range
byte	Unsigned 8-bit integer	0 to 255
sbyte	Signed 8-bit integer	-128 to 127
short	Signed 16-bit integer	-32,768 to 32,767
ushort	Unsigned 16-bit integer	0 to 65,535
int	Signed 32-bit integer	-2,147,483,648 to 2,147,483,647
uint	Unsigned 32-bit integer	0 to 4,294,967,295
long	Signed 64-bit integer	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
ulong	Unsigned 64-bit integer	0 to 18,446,744,073,709,551,615
float	32-bit floating point with 7 digits precision	±1.5e−45 to ±3.4e38
double	64-bit floating point with 15-16 digits precision	±5.0e−324 to ±1.7e308
decimal	128-bit floating point with 28-29 digits precision	±1.0 × 10e−28 to ±7.9 × 10e28
char	Unicode 16-bit character	U+0000 to U+ffff
bool	Stores either true or false	true or false

Reference Types

Object	• Object is a built-in reference data type that is a base class for all pre-defined and user-defined data types.
String	• String is a built-in reference type that signifies Unicode character string values.
Class	• A class is a user-defined structure that contains variables and methods.
Delegate	• A delegate is a user-defined reference type that stores the reference of one or more methods.
Interface	• An interface is a user-defined structure that groups related functionalities which may belong to any class or struct.
Array	• An array is a user-defined data structure that contains values of the same data type, such as marks of students.

Declaration

- In C#, you can declare multiple variables at the same time in the same way you declare a single variable.
- After declaring variables, you need to assign values to them.
- Assigning a value to a variable is called initialization.
- You can assign a value to a variable while declaring it or at a later time.

Syntax

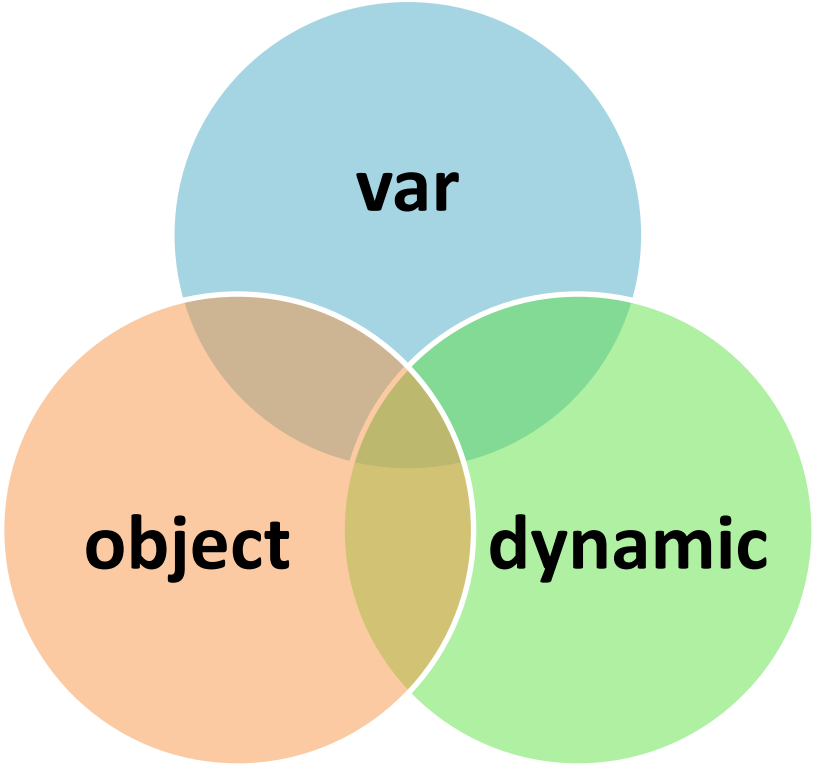
- <data type> <variable name> [= <value>];
  - ✓ data type: is a valid variable type.
  - ✓ variable name: is a valid variable name or identifier.
  - ✓ value (optional): is the value assigned to the variable.

- In C#, you can decide to declare and assign value or declare variable only
- In C#, you can declare multiple variables have same data type at the time

Data Type in practice

- Create new project and give variables to:
  - ✓ Store name of user
  - ✓ Store age of student
  - ✓ Store weight of elephant
  - ✓ Store GDP of United States
  - ✓ Store test result of Corona virus quick test

Comparison



Variable name

- The name can contain letters, digits, and the underscore character (_).
- The first character of the name must be a letter. The underscore is also a legal first character, but its use is not recommended at the beginning of a name. An underscore is often used with special commands, and it's sometimes hard to read.
- Case matters (that is, upper- and lowercase letters). C# is case-sensitive; thus, the names count and Count refer to two different variables.
- C# keywords can't be used as variable names. Recall that a keyword is a word that is part of the C# language.

Variable name example

Variable Name	Legality
Percent	Legal
y2x5__w7h3	Legal
yearly_cost	Legal
_2010_tax	Legal, but not advised
checking#account	Illegal; contains the illegal character #
double	Illegal; is a C keyword
2towers	Illegal; starts with number

Naming conventions

- **Use Camel Casing** - First character of all words, except the first word are Upper Case and other characters are lower case.
  - ✓ Example: backColor
- Do not use Hungarian notation to name variables
  - ~~✓ string m_sName;~~
- Use Meaningful, descriptive words to name variables. Do not use abbreviations
  - ✓ Good: userAddress
  - ~~✓ Bad: uAdd~~
- **Re-check and update your code**

Comments

- Comments help in reading the code of a program to understand the functionality of the program.
- Comments are ignored by the compiler, during the execution of the program.

Comments

- Provide information about a piece of code.
- Make the program more readable.
- Explain the purpose of using a particular variable or method to a programmer.
- Help to identify comments as they are marked with special characters.

Comments

- Single-line Comments: Begin with two forward slashes (//).
- Multi-line Comments: Begin with a forward slash followed by an asterisk (/*) and end with an asterisk followed by a forward slash (*/).
- **Re-check and update your code**

Constants

- A constant has a fixed value that remains unchanged throughout the program while a literal provides a mean of expressing specific values in a program.
- Examples:
  - ✓ the ratio of the circumference of a circle to its diameter:
    - π = 3.1415926535897931
    - Provided in Math.PI
  - ✓ the natural logarithmic base:
    - e = 2.718281828590451
    - Provided in Math.E

Constants

- In C#, you can declare constants for all data types.
- You have to initialize a constant at the time of its declaration.
- Constants are declared for value types rather than for reference types.
- To declare an identifier as a constant, the const keyword is used in the identifier declaration. The compiler can identify constants at the time of compilation, because of the const keyword.

Constant syntax

- const <data type> <identifier name> = <value>;
  - ✓ const: Keyword denoting that the identifier is declared as constant.
  - ✓ data type: Data type of constant.
  - ✓ identifier name: Name of the identifier that will hold the constant.
  - ✓ value: Fixed value that remains unchanged throughout the execution of the code

Section 2

KEYWORDS

Keywords

- Keywords are reserved words that are separately compiled by the compiler and convey a pre-defined meaning to the compiler and hence, cannot be created or modified.
  - ✓ Example: int, double, string, break, var, …
- Escape sequence characters in C# are characters preceded by a backslash (\) and denote a special meaning to the compiler.
- You cannot use keywords as variable names, method names, or class names, unless you prefix the keywords with the '@' character

C# keywords

- Keywords in C# is mainly divided into 10 categories:
  1. Value Type Keywords
  2. Reference Type Keywords
  3. Modifiers Keywords
  4. Statements Keywords
  5. Method Parameters Keywords
  6. Namespace Keywords
  7. Operator Keywords
  8. Conversion Keywords
  9. Access Keywords
  10. Literal Keywords

Value Type keywords

- There are **15** keywords in value types which are used to define various data types.

bool	byte	char	decimal
double	enum	float	int
long	sbyte	short	struct
unit	ulong	ushort	

Reference Type Keywords

- There are 6 keywords in reference types which are used to store references of the data or objects.
- The keywords in this category are:

class	delegate	interface
object	string	void

Modifiers Keywords

- There are **17** keywords in modifiers which are used to modify the declarations of type member.

public	private	internal	protected	abstract
const	event	extern	new	override
partial	readonly	sealed	static	unsafe
virtual	volatile			

Statements Keywords

- There are total **18** keywords which are used in program instructions

if	else	switch	do	for
foreach	in	while	break	continue
goto	return	throw	try	catch
finally	checked	unchecked		

Others Keywords

- There are total **4** keywords which are used to change the behaviour of the parameters that passed to a method: params, in, ref, out
- There are total 3 keywords in this category which are used in namespaces: namespace, using, extern
- There are total 8 keywords which are used for different purposes like creating objects, getting a size of object etc…: as, is, new, sizeof, typeof, true, false, stackalloc.

Others Keywords

- There are 3 keywords which are used in type conversions. The keywords are: explicit, implicit, operator.
- There are 2 keywords which are used in accessing and referencing the class or instance of the class. The keywords are base, this.
- There are 2 keywords which are used as literal or constant. The keywords are null, default.

Important Points

- Keywords are not used as an identifier or name of a class, variable, etc.
- If you want to use a keyword as an identifier then you must use @ as a prefix.
  - ✓ For example, @abstract is valid identifier but not abstract because it is a keyword.

Contextual keywords

- C# provides contextual keywords that have special meaning in the context of the code, where they are used.
- The contextual keywords are not reserved and can be used as identifiers outside the context of the code.
- When new keywords are added to C#, they are added as contextual keywords.

Contextual keywords

add	alias	ascending	async	await
by	descending	dynamic	equals	from
get	global	group	into	join
let	nameof	on	orderby	partial(type)
partial(method)		remove	select	set
value	var	yield	when(filter condition)	
where (generic type constraint)			where (query clause)	
unmanaged (generic type constraint)				

Important Points

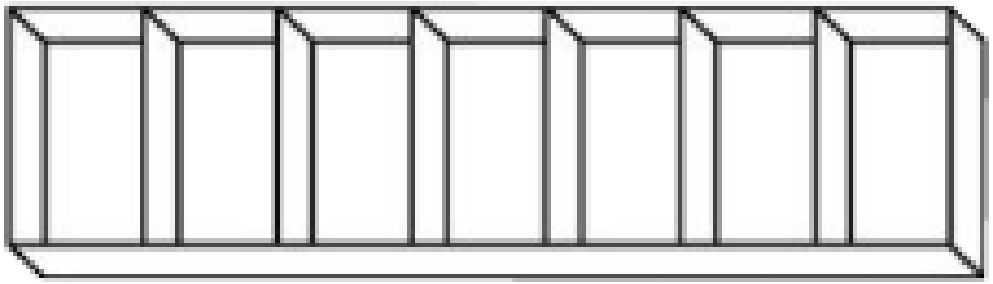
- These are not reserved words.
- It can be used as identifiers outside the context that's why it named contextual keywords.
- These can have different meanings in two or more contexts

Section 3

ARRAY

Introduction

- An array is a collection of elements of a single data type stored in adjacent memory locations.
- Example:
  - ✓ In a program, an array can be defined to contain 7 elements to store the scores of 7 students.



Purpose of array

- Consider a program that stores the names of 100 students.
- To store the names, the programmer would create 100 variables of type string.
- Creating and managing these 100 variables is a tedious task as it results in inefficient memory utilization.
- In such situations, the programmer can create an array for storing the 100 names.

Array of 100 Names

Steve	David	John	Klen	Stefen	.....

Proper Utilization of Memory

100 Variables Storing Names

Program to store 100 names of students	
var empOne	Steve
var studentTwo	David
var studentThree	John
var studentFour	Klen
var studentFive	Stefen
…	…
… Till 100 variables	

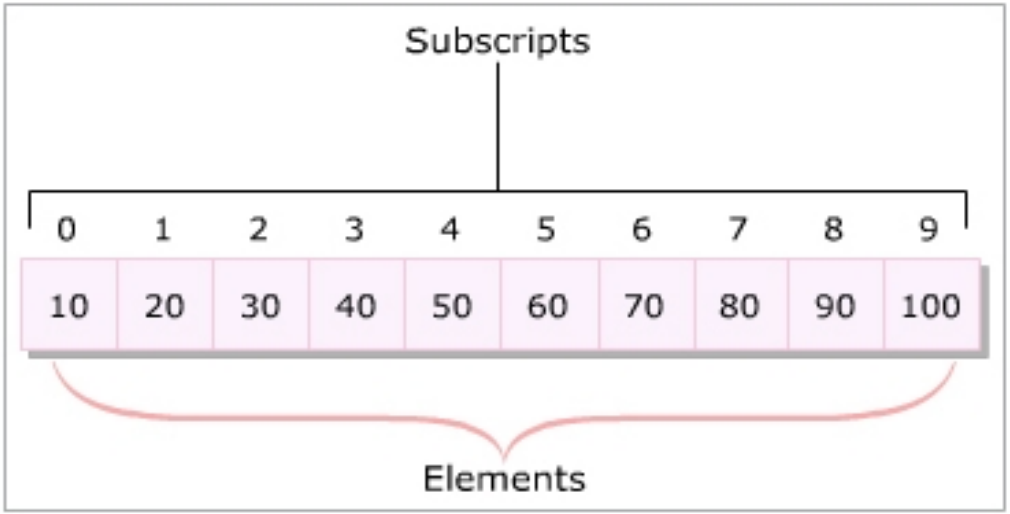
Inefficient Memory Utilization

Array

- An array always stores values of a single data type.
  - ✓ Each value is referred to as an element.
- These elements are accessed using subscripts or index numbers that determine the position of the element in the array list.
- C# supports zero-based index values in an array.
  - ✓ This means that the first array element has an index number zero while the last element has an index number n-1, where n stands for the total number of elements in the array.
- This arrangement of storing values helps in efficient storage of data, easy sorting of data, and easy tracking of the data length.

Array

- Following figure is an example of the subscripts and elements in an array:



Arrays

Declaring Array

- Arrays are reference type variables whose creation involves two steps:
  - ✓ **Declaration:**
    - An array declaration specifies the type of data that it can hold and an identifier.
    - This identifier is basically an array name and is used with a subscript to retrieve or set the data value at that location.
  - ✓ **Memory allocation:**
    - Declaring an array does not allocate memory to the array.

Declaring Array

- Following is the syntax for declaring an array:

  <type>[] <array name>

- In the syntax:
  - ✓ type: Specifies the data type of the array elements (for example, int or char).
  - ✓ array name: Specifies the name of the array.
- Example:

```
int[] studentScores;
```

Initializing array

- An array can be:
  - ✓ Created using the `new` keyword and then initialized.
  - ✓ Initialized at the time of declaration itself, in which case the `new` keyword is not used.
- Creating and initializing an array with the `new` keyword involves specifying the size of an array.
- The number of elements stored in an array depends upon the specified size.

Initializing array

- The new keyword allocates memory to the array and values can then be assigned to the array.
- If the elements are not explicitly assigned, default values are stored in the array.

Initializing array

- The following syntax is used to initialize an array:

  <array name> = new type[size-value];

- Or declare and initialize at the time:

  type[] <array name> = new type[size-value];

  var <array name> = new type[size-value];

- In the syntax:
  - ✓ `size-value`: Specifies the number of elements in the array.
  - ✓ You can specify a variable of type `int` that stores the size of the array instead of directly specifying a value.

Initializing array

- Examples:

  studentScores = new int[30];

  topTenProducts = new string[10];

  ….

- Important!
  - ✓ Size of array CANNOT change after the array initialize
  - ✓ To make sure you choose appropriate size for the array

Initializing elements

- Once an array has been created using the syntax, its elements can be assigned values using either a subscript or index
- Example:
  - ✓ topTenProducts[0] = “Macbook Pro”; //// the 1st product
  - ✓ topTenProducts[3] = “Dell Latitute”; //// the 4th product
  - ✓ topTenProducts[10] = “HP Compact”; //// exception

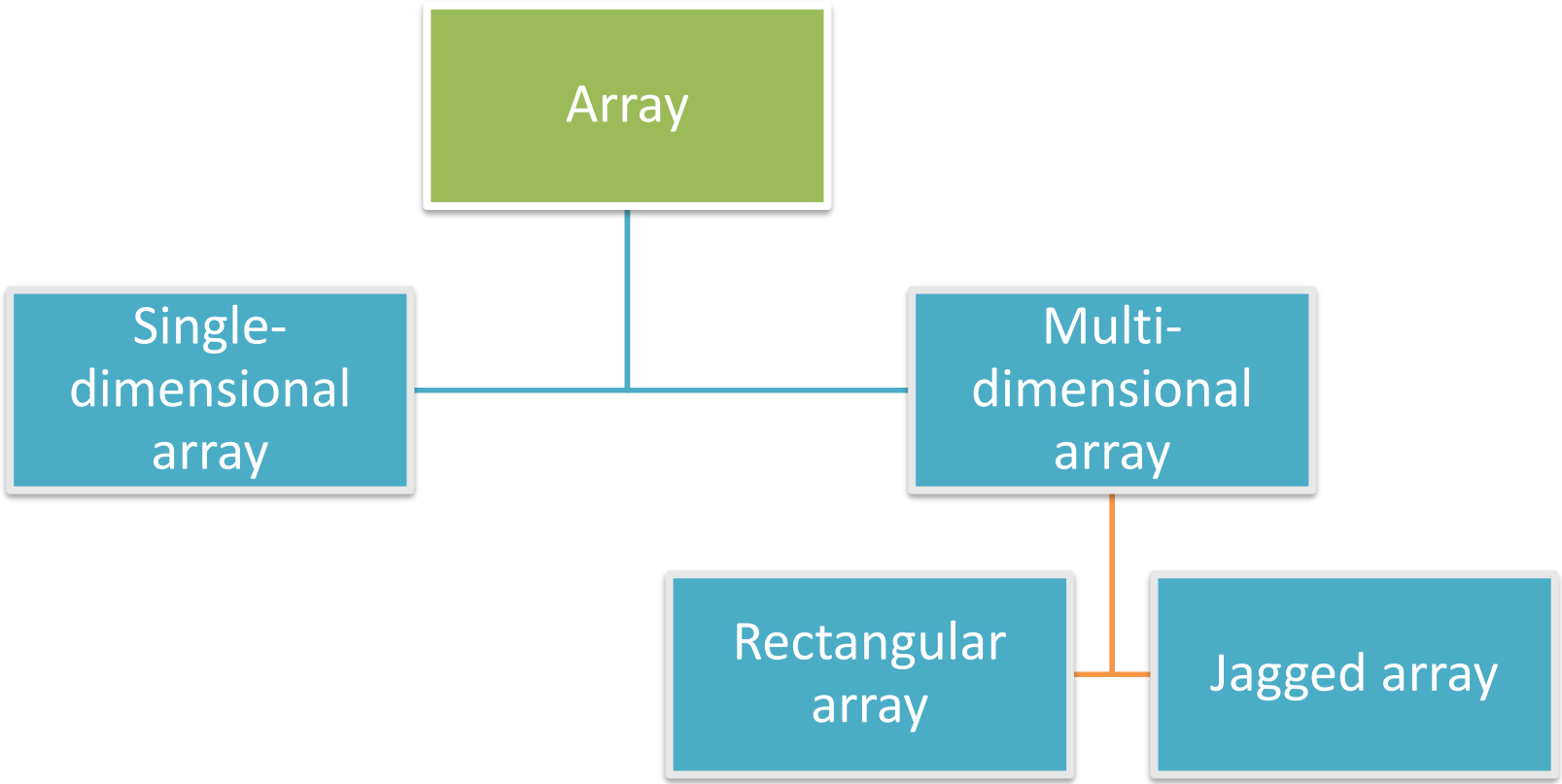
Initializing elements

- Others:
  - ✓ topProducts = new string[3] { "Macbook Pro", "Dell Latitute", "HP Compact" };
    - Question: can we make it shorter?
  - ✓ topProducts = new string[] { "Macbook Pro", "Dell Latitute", "HP Compact" };
    - Question: size of array?
  - ✓ string[] topProducts = { "Macbook Pro", "Dell Latitute", "HP Compact" };
    - Question: can we use var in this case?

Get element from array

- Use index to access and get element
- Random access to any element in array
- Always check the boundary of array to avoid exception

More type of array



Rectangular array

- A rectangular array is a n-dimensional array where size of each dimension are equal.
- It usually used to represent matrix. It's very important to help to resolve math problems.
- Example:
  - ✓ Linear Transformations
  - ✓ Matrix-Vector Multiplication Algorithms
  - ✓ Gaussian Elimination
  - ✓ LU Factorization
  - ✓ The Inverse Matrix

Rectangular array

- Syntax:
  - ✓ type [,]<variableName>;
  - ✓ variableName = new type[value1 , value2];
- In the syntax:
  - ✓ type: Specifies the data type of the array elements.
  - ✓ [,]: Specifies that the array is a two-dimensional array.
  - ✓ value1: Specifies the number of rows in the two-dimensional array.
  - ✓ value2: Specifies the number of columns in the two-dimensional array.

Rectangular array

- Assign and get element: based on index with dimension in order
- Example:
  - ✓ Create a 2-dimensional array to store the matrix:

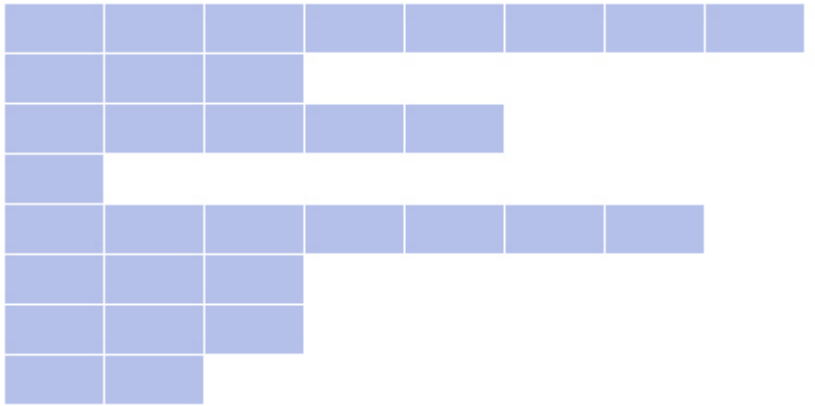
4	2	5	5	7
10	5	-3	0	1
5	11	-4	9	6
3	-8	4	12	0

Jagged array

- Is a multi-dimensional array and is referred to as an array of arrays.
- Consists of multiple arrays where the number of elements within each array can be different. Thus, rows of jagged arrays can have different number of columns.
- Optimizes the memory utilization and performance because navigating and accessing elements in a jagged array is quicker as compared to other multi-dimensional arrays.

Example

- Consider a class of 50 students where each student has opted for a different number of subjects.
- Here, you can create a jagged array because the number of subjects for each student varies.
- The following figure displays the representation of jagged arrays:



Jagged array

- Syntax and example:
  - ✓ Use [][] instead of [,]
  - ✓ Declare size one-by-one

```
//// Class has 50 students
decimal[][] studentSubjects = new decimal[50][];

//// 1st student has 15 subjects
studentSubjects[0] = new decimal[15];

//// Assign (and get) element
studentSubjects[0][0] = 4.0m;
studentSubjects[0][9] = 7.8m;
```

Thank you