ARRAYLIST IN C++ WITH EXAMPLES

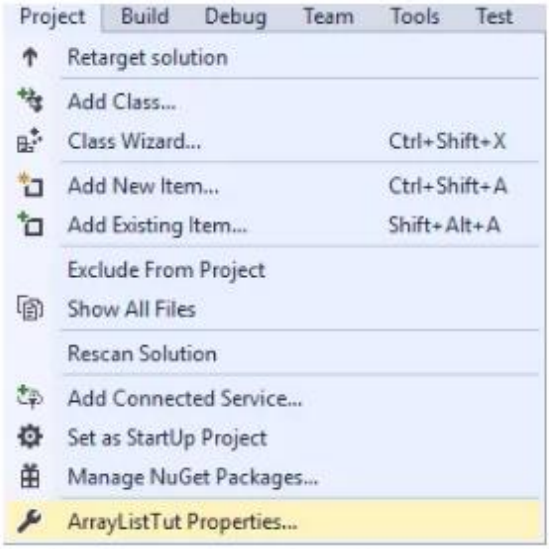
*ArrayList*s have been used by programmers since the late 90's. They were implemented to be more flexible collections than arrays. They are implemented in many programming languages such as Java, C# and C++. *ArrayLists* are a type of collection that can be used to store various types of data. This means two different classes like string and integer can be stored together in a single collection. There are both advantages and disadvantages to *ArrayLists.*

An *ArrayList* is a flexible list of objects that can be dynamically resized and accessed through Integer indexing. It can store many types of variable except for a multi-dimensional array. This includes null values. Unfortunately *ArrayLists* are inefficient compared to arrays and Generic Lists. *ArrayList*s are not type-safe, and therefore must be boxed, and unboxed otherwise an error at runtime will occur. *ArrayList*s are not thread-safe by default.

Due to the lack of type-safety and poor performance when using *ArrayList*s they have been replaced throughout the years in C++ by usin Lists have all the same benefits as *ArrayList*s. Generic Lists are type-safe more reliable, more bug-free and recommended to use. *ArrayList*s shou legacy code bases.

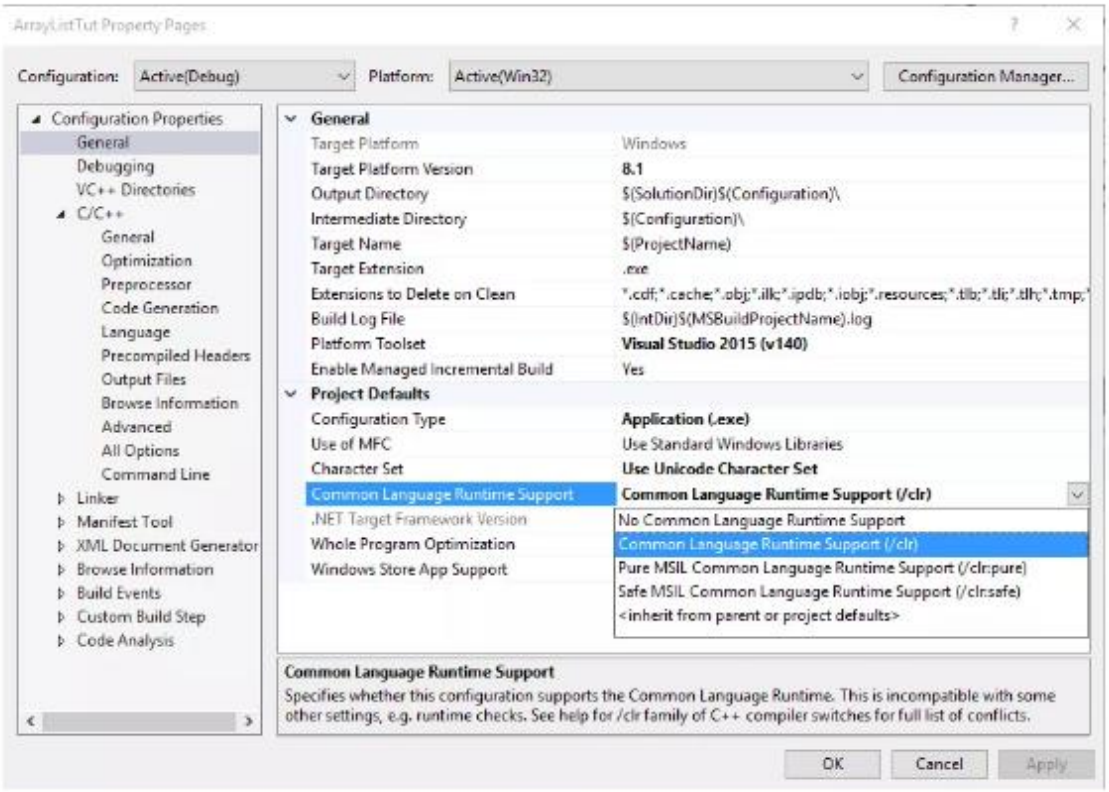
ArrayList C++ Implementation:-

ArrayLists are not in the standard library in C++. *ArrayLists* and Generic Lists are compliant with the Common Language Runtime also known as the CLR. To use *ArrayLists* and Generic Lists you must enable the CLR in your solution's project files.



To do this open Project>Project Name Properties… in the toolbar.

Inside the Configuration Properties open the General tab. Under Project Defaults find the row Common Language Runtime Support. Select Common Language Runtime Support (/clr) from the drop down menu.



Do the same for Configuration Properties -> C/C++ to fully enable the Common Language Runtime. After this you are free to use *ArrayLists* to store heterogeneous data in your solution.

ArrayList C++ Properties:-

Capacity — The capacity property represents how many elements the *ArrayList* can store.

Count — The count property represents how many elements the *ArrayList* currently has.

IsFixedSize — Does the *ArrayList* have a fixed size?

IsReadOnly — Is the *ArrayList* readonly?

IsSynchronized — Is the *ArrayList* thread-safe?

Item[Integer] — The element of the *ArrayList* given at the index.

SyncRoot — It is used to store a synchronized value of the *ArrayList.*

ArrayList C++ Methods:-

Adapter(IList^) — Wrap an object with the interface IList^ as an *ArrayList.*

Add(Object^) — Add a value to the end of the *ArrayList.*

AddRange(ICollection^) — Adds a range of elements to the end of the ArrayList.

BinarySearch(Object^) — Sort the ArrayList for an Object.

Clear() — Empties the ArrayList.

Clone() — Duplicate the ArrayList.

Contains(Object^) — Check if the ArrayList has the element given.

CopyTo(Array^) — Casts the ArrayList into an Array.

Equals(Object^) — Check if the ArrayList is equal to another ArrayList.

Finalize() — Frees excess resources being used by the ArrayList.

FixedSize() — Returns the ArrayList with a fixed size.

GetEnumerator() — Casts the ArrayList as an Enumerator.

GetHashCode() — Hashes the ArrayList.

GetRange(Int, Int) — Returns the elements starting at the first index and ending at the second index provided.

GetType() — Returns the type ArrayList.

IndexOf(Object^) — Returns the index of the element that contains the object given.

Insert(Int, Object^) — Adds an element to the list at the given index.

InsertRange(Int, ICollection^) — Adds a range of elements at the given index.

LastIndexOf(Object^) — Returns the index of the last occurrence of an object.

MemberwiseClone() — Creates a copy of the base class Object.

ReadOnly(ArrayList^) — Returns the ArrayList as read-only.

Remove(Object^) — Removes the element the first time it is found.

RemoveAt(Int) — Removes the element at the given index.

RemoveRange(Int, Int) — Removes the elements starting at the first index and ending at the last index.

Repeat(Object^, Int) — Create a new ArrayList with a value that repeats itself the given int amount of times.

Reverse() — Reverse the order of the elements in the ArrayList.

SetRange(Int, ICollection^) — Copy the elements of a collection into an ArrayList at a given index.

Sort() — Sorts the elements of the ArrayList by alphabetical order.

Synchronized(ArrayList^) — Returns a thread-safe version of the ArrayList.

ToArray() — Returns the ArrayList as an Object array,

ToString() — Returns the ArrayList type as a string.

TrimToSize() — Reduces the capacity to the amount of elements in the ArrayList

ArrayList C++ Example:-

// ArrayListTut.cpp : Defines the entry point for the console application.

//

```
#include <iostream>
#include "stdafx.h"
using namespace std; //Import the standard library
using namespace System; //Using the System Library
using namespace System::Collections; //Using the System Collections Library
int main()
{
ArrayList^ ArrayList = gcnew ArrayList; //Initialize a new ArrayList
//Add various values to the ArrayList
ArrayList->Add("Bob Ross"); //Add value to the end of the ArrayList
ArrayList->Add("Bob Ross"); //Add duplicate value
ArrayList->Add(216); //Add integer
ArrayList->Add(43.1f); //Add floating point
//Allow for enumeration of the ArrayList
IEnumerator^ enumerator = ArrayList->GetEnumerator();
//Iterate through each value in the ArrayList
while (enumerator->MoveNext())
{
//Cast the object into its corresponding type
Object^ object = safe_cast<Object^>(enumerator->Current);
//Print the type followed by the value of the variable
Console::WriteLine("Type: " + object->GetType() + " Value: " + object);
}
//Wait for input
Console::ReadLine();
}
```

Explanation of ArrayList C++ Example:-

The above example includes *<iostream>* and *"stdafx.h"*. This allows for the use of the standard library and utilize standard types. The three using namespaces are used to shorten the lines of code.

The ArrayList type contains '^' and gcnew. This is necessary for the managed code to be properly allocated to memory. In the next four lines ArrayList is added to using ArrayList->Add(value). This demonstrates that ArrayLists can contain duplicates and heterogenous variables.

The ArrayList is then explicitly cast as the type Enumerator. This is to allow the enumeration in the following while loop below to iterate through each element in the ArrayList.

Inside the while loop the elements of the ArrayList are casted into their corresponding object types. The objects are then printed as the type of object, followed by the value of the object.

The program is forced to wait at Console::ReadLine until any button is pressed to close the application.