Adnans C# windows application learning start

Basic syntax:

First you need to start with an idea, the idea is a server application that uses discord.

WindowsGSM will print messages to the discord if the server is offline and when it comes online, also it can send out messages every 15 – 1hr to the discord to tell the discord members that the server is still running properly.

I will be allowing you to borrow my Computer science book to learn the basics of C++ so you can learn C# as well, this is the purpose of this documentation of lesson 01.

So far the first step is configuring the User interface. For this we use the coding language (C#).

This coding language uses different syntax compared to C++ for example in C++ you the programmer would use #include<iostream> for this output (cout << “Hello world” << endl;) this also equals (std::cout << “Hello world” << std::endl;), don’t for get when using the short hand you will also need to include using namespace std; , or else you will have to use option 2.

In C# you as the programmer will use (using (name of library; , forget about the () )), as for changing the syntax for output fields in C# there is none.

At the start of the file C++ and C# are completely different syntax for C++ its int main(){} as a C# programmer you will use the namespace YourNameSpace (name you want to call the file);

Next are variables: in C++ the variables are Bool, int 45324235423645643, float 1.50982340763249876, longint, longlongint, double 24.46, longdouble, , string “hello world” (can save use white space), char ‘C’ (cant use white space) and ASCII values for A-Z.

As a C# programmer you will be using these variables: int 454685678, float 1.5f, string “Hello world”, char ‘C’, Bool and ASCII values A-Z

The list below is an example of all the peramerters used in C++ and C#

**C++:**

1. **Primitive Types:**
   * **int**: Integer data type.
   * **float**: Single-precision floating-point data type.
   * **double**: Double-precision floating-point data type.
   * **char**: Character data type.
   * **bool**: Boolean data type.
2. **Modifiers:**
   * **signed** and **unsigned**: Modifiers for integer types to indicate signed or unsigned representation.
   * **short** and **long**: Modifiers for integer types to specify size.
3. **Derived Types:**
   * **arrays**: A collection of elements of the same data type.
   * **struct**: User-defined composite data type.
   * **enum**: Enumeration types for defining named constants.
   * **class**: User-defined data type that can include member functions and variables.
4. **Pointers:**
   * **\***: Pointer declaration.
   * **&**: Reference operator.
   * **nullptr**: Null pointer literal.

**C#:**

1. **Primitive Types:**
   * **int**: 32-bit signed integer.
   * **float**: 32-bit single-precision floating-point number.
   * **double**: 64-bit double-precision floating-point number.
   * **char**: 16-bit Unicode character.
   * **bool**: Boolean data type.
2. **Modifiers:**
   * **uint**: 32-bit unsigned integer.
   * **short** and **long**: 16-bit and 64-bit signed integer respectively.
3. **Derived Types:**
   * **arrays**: Similar to C++, arrays store a collection of elements of the same data type.
   * **struct**: Similar to C++, structs are lightweight data structures that can contain variables and methods.
   * **enum**: Similar to C++, enumeration types allow you to define named constants.
   * **class**: Fundamental building block of C# programs, allowing you to create objects with methods and properties.
   * **interface**: Defines a contract that classes can implement.
   * **delegate**: Similar to function pointers in C++, delegates are used to create type-safe function pointers.
   * **record**: Introduced in C# 9.0, a lightweight alternative to classes for immutable data.
4. **Pointers:**
   * Pointers in C# are managed by the runtime and are not directly accessible or manipulable by developers in the same way as in C++. However, you can use pointers in "unsafe" contexts in C# using the **unsafe** keyword, but this is rarely used in typical C# development.

When making a windows application you have public and private. Public can be accessed threw out the whole class and private can only be used and accessed within the class.

C++ syntax:

**Public Variables:**

* Public variables in C++ classes are accessible from outside the class.
* They can be accessed directly by objects of the class or through class member functions.
* Public variables are often used when you need direct access to the data members of the class.

Class MyClass {

Public:

Int APublicIntVar;

}

**Private Variables:**

* Private variables in C++ classes are only accessible within the class itself.
* They cannot be accessed directly from outside the class.
* Access to private members is typically controlled through public member functions, often referred to as getter and setter methods.

Class MySecoundClass{

Private:

Int MyPrivateInt;

Public:

Void setPrivateVar(int value){

privateVar = value;

}

Int GetPrivateVar(){

Return privateVar;

}

}

C# syntax:

**Public Variables:**

* Public variables in C# classes are accessible from outside the class.
* They follow similar rules as in C++ and can be accessed directly by objects of the class or through class member functions.
* Public variables are typically used when you need direct access to the data members of the class.

Public class MyClass{

Public int PublicVar;

}

**Private Variables:**

* Private variables in C# classes are only accessible within the class itself.
* They cannot be accessed directly from outside the class.
* Access to private members is controlled through public properties or methods, similar to C++.

Public class mySecoundClass{

Private int MyPrivateVar;

Public int PrivateVar{

Get {return privateVar;}

Get{privateVar = value;}

}

}

Next, as a C# programmer you will need to understand how to use ENUM (enumerations).

C++ syntax:

In C++, enums by default start from 0, and subsequent enumerators increment by 1. You can explicitly specify values for the enumerators:

Enum Color{

Red,

Green,

Blue

};

Enum ColorTwo{

Red = 10,

Green = 20,

Blue = 30

};

C# syntax:

Public enum color{

Red,

Green

blue

} (no need for the (;))

In C#, enums are strongly typed and can't be implicitly converted to integer types. If you want to explicitly set the underlying values of the enum members, you can do it like this:

Color myColor = Color.red;

If(myColor == Color.red){

}

Public enum SecoundColor {

Red = 10,

Green = 20,

Blue = 30

}

Additional points:

During this portion you will be learning how to use Scoped enums.

C++ syntax:

**Scoped Enums (C++):** In C++, you can use scoped enums (enum class) to prevent name clashes. Scoped enums require the use of the enum's type to access its enumerators.

Enum class Status{

OK,

Error

};

Status myStatus = Status::OK;

C# syntax:

**Flags Enums (C#):** In C#, you can use enums with the **[Flags]** attribute to create bit flags enumeration. This allows combining multiple values using bitwise OR.

[Flags]

Public Enum DaysOfWeek{

None = 0,

Monday = 1,

Tuesday = 2,

Wednesday = 3,

Thursday = 4,

Friday = 5,

Saturday = 6,

Sunday = 7

}

DaysOfWeek MyDays = DaysOfWeek.Monday | DaysOfWeek.Thursday;

Next you will learn about the use of auto in C++, auto is the same as using bool,int,float,double,char the difference is that auto is using more recourses from the computer at startup time to decide what type of variable you are trying to initialize and the fact that (var y = “hello” is a const char). In C# you will replace auto with var and it functions the same way, except when you use the var (“Hello”, it actually is a string unlike in C++). The best example for using auto is when you need a variable for hyperLink, stackPannel, authorImg.

Next you will learn about the Try method. The try method is used to verify code, you know like try to do this (get server IP to see if the server is online).

In C++, the **try** block is used to enclose a block of code that may throw an exception. It is followed by one or more **catch** blocks that handle specific types of exceptions. Additionally, C++ allows for a **finally** block

C++ syntax:

try {

// Code that may throw an exception

throw SomeException();

}

catch (const SomeException& e) {

// Handle the specific exception

}

catch (const AnotherException& e) {

// Handle another specific exception

}

// Optionally, a finally block for cleanup

finally {

// Cleanup code

}

C# syntax:

try {

// Code that may throw an exception

throw new SomeException();

}

catch (SomeException e) {

// Handle the specific exception

}

catch (AnotherException e) {

// Handle another specific exception

}

finally {

// Cleanup code or code that must execute regardless of exceptions

}

In C#, the **try** block is also used to enclose a block of code that may throw an exception. It is followed by one or more **catch** blocks that handle specific types of exceptions, similar to C++. However, C# does not have a **finally** block. Instead, it has a **finally** keyword that is used with **try** blocks to ensure certain code executes, regardless of whether an exception is thrown or not. The **finally** block executes after any matching **catch** blocks, and even if there is a **return** statement in the **try** or **catch** block.

**await** is a keyword used in asynchronous programming in both C++ and C#, but there are significant differences in how it's implemented and used in each language.

**C++:**

In C++, **await** is part of the C++/WinRT library, which is used for asynchronous programming with Windows Runtime (WinRT) components. It's used to asynchronously wait for the result of a task or operation that returns a **Windows::Foundation::IAsyncOperation** or similar asynchronous types.

Example:

// Assuming an asynchronous operation returns IAsyncOperation<TResult>

auto result = co\_await SomeAsyncOperation();

**await** is a keyword used in asynchronous programming in both C++ and C#, but there are significant differences in how it's implemented and used in each language.

**C++:**

In C++, **await** is part of the C++/WinRT library, which is used for asynchronous programming with Windows Runtime (WinRT) components. It's used to asynchronously wait for the result of a task or operation that returns a **Windows::Foundation::IAsyncOperation** or similar asynchronous types.

Example:

cppCopy code

// Assuming an asynchronous operation returns IAsyncOperation<TResult> auto result = co\_await SomeAsyncOperation();

In this example, **co\_await** is used to asynchronously wait for the result of the **SomeAsyncOperation**.

**C#:**

In C#, **await** is a language feature introduced in C# 5.0 for asynchronous programming with the **async** and **await** keywords. It's used to asynchronously wait for the completion of an asynchronous operation that returns a **Task** or **Task<TResult>**.

Example:

// Assuming an asynchronous method returns Task<TResult>

var result = await SomeAsyncMethod();

next you will learn about the foreach statement in C++ and C#. this statement is used for more cleaner and more readable code.

C++ syntax:

std::vector<int> numbers = {1, 2, 3, 4, 5};

// Iterate over each element in the vector

for (int num : numbers) {

// Access num

std::cout << num << std::endl;

}

C# syntax:

int[] numbers = {1, 2, 3, 4, 5};

// Iterate over each element in the array

foreach (int num in numbers) {

// Access num

Console.WriteLine(num);

}

Next you will learn about the switch statement in C++ and C#. the switch statement is used as shorthand, it replaces the if statements.

C++ syntax:

int num = 2;

switch (num) {

case 1:

// Code block for case 1

break;

case 2:

// Code block for case 2

break;

default:

// Code block for default case

break;

}

C# syntax:

int num = 2;

switch (num) {

case 1:

// Code block for case 1

break;

case 2:

// Code block for case 2

break;

default:

// Code block for default case

break;

}

1. **Supported Types:**
   * In C++, **switch** statements primarily work with integral types like **int**, **char**, **enum**, etc. It doesn't support string comparisons directly.
   * In C#, **switch** statements support a wider range of types, including integral types, strings, enums, and other types that can be compared for equality.
2. **Fallthrough Behavior:**
   * In C++, **switch** statements have fallthrough behavior by default, meaning if a **case** label is matched, the execution continues to subsequent **case** labels unless a **break** statement is encountered.
   * In C#, fallthrough behavior is not allowed. Once a **case** label is matched, the execution stops unless a **break** statement is encountered.
3. **Pattern Matching (C#):**
   * C# introduces pattern matching capabilities in **switch** statements, allowing for more complex matching scenarios using **case when** clauses.
4. **Default Case:**
   * Both C++ and C# support a **default** case that executes when none of the **case** labels match the expression.
5. **Enum Support (C#):**
   * C# allows using enums directly in **switch** statements without casting to integral types, which can improve code readability and safety.

In both C++ and C#, **async** is a keyword used in asynchronous programming to define methods that can execute asynchronously. However, the implementation and usage of **async** differ between the two languages.

**C++:**

In C++, asynchronous programming is typically done using the **std::future**, **std::async**, and related standard library components introduced in C++11 and later. The **async** keyword itself is not part of the C++ language syntax.

However, in modern C++ programming, you can use the **co\_await** and **co\_return** keywords with coroutines, introduced in C++20, to write asynchronous code more concisely. Coroutines enable you to suspend and resume execution of a function at specific points, allowing for more readable asynchronous code.

Here's a simplified example using coroutines in C++:

#include <iostream>

#include <experimental/coroutine>

struct MyTask {

struct promise\_type {

MyTask get\_return\_object() { return {}; }

std::experimental::suspend\_never initial\_suspend() { return {}; }

std::experimental::suspend\_never final\_suspend() noexcept { return {}; }

void return\_void() {}

void unhandled\_exception() {}

};

};

MyTask MyAsyncMethod() {

// Asynchronous operation

co\_await std::experimental::suspend\_always{};

std::cout << "Async operation completed." << std::endl;

}

int main() {

MyAsyncMethod();

std::cout << "Main function continues execution." << std::endl;

return 0;

}

In this example, **MyAsyncMethod** is an asynchronous function that suspends execution using **co\_await** and then resumes execution when the asynchronous operation completes.

**C#:**

In C#, the **async** keyword is used to define asynchronous methods that can execute asynchronously. These methods typically return a **Task** or **Task<TResult>** object representing an asynchronous operation that may complete at a later time.

Here's an example of using **async** and **await** in C#:

using System;

using System.Threading.Tasks;

class Program

{

static async Task MyAsyncMethod()

{

// Asynchronous operation

await Task.Delay(1000);

Console.WriteLine("Async operation completed.");

}

static async Task Main(string[] args)

{

await MyAsyncMethod();

Console.WriteLine("Main function continues execution.");

}

}

In this example, **MyAsyncMethod** is an asynchronous method that simulates an asynchronous operation using **Task.Delay**. The **await** keyword is used to asynchronously wait for the completion of the asynchronous operation.

Next we will learn about the while statement. This statement is the same in C++ and C#:

C++ and C# syntax:

int i = 0;

while (i < 5)

{

std::cout << i << std::endl;

i++;

}