CERTIFIED QUIZ

ntroduction to C#

C# (pronounced 'C-sharp') is a modern, object-oriented, and type-safe programming language developed by Microsoft as part of its .NET initiative. It was designed by Anders Hejlsberg and first released in 2002. C# is widely used for developing a range of applications, including web, desktop, and mobile applications. It is statically typed, meaning type checking is performed at compile-time. C# is syntactically similar to Java, C++, and other languages in the C family.

Structure of a C# Program

A typical C# program consists of the following parts:
- Namespace Declaration: Defines a named scope for classes and other code elements.
- Class Declaration: Defines the blueprint of an object with members like methods and properties.
- Main Method: The entry point for program execution, defined as `static void Main()`.
- Statements and Expressions: Perform operations and computations.

Example:
```
csharp
using System;

namespace HelloWorld {
class Program {
static void Main(string[] args) {
Console.WriteLine("Hello, World!");
}
}
}
```

Data Types in C#

C# offers a wide variety of data types, categorized into Value Types and Reference Types:

- Value Types: Store data directly.
- `int`: Represents integers.
- `float`: Represents floating-point numbers.
- `bool`: Represents Boolean values (`true` or `false`).
- `char`: Represents a single character.
- `struct`: A value type that groups related variables.

- Reference Types: Store a reference to the memory location where data is stored.
- `string`: Represents a sequence of characters.
- `object`: The base type from which all other types are derived.
- `class`: Represents an object with fields and methods.

Example:
```
csharp
int age = 25;
float pi = 3.14f;
bool isActive = true;
char initial = 'A';
string name = "John";
```

Variables and Constants

Variables store data that can change during program execution. They are declared using a specific data type. Variables must be initialized before use.

Example:
csharp
```
int age = 30;
float temperature = 98.6f;
```

Constants are fixed values that cannot be modified after their initial assignment. Constants are defined using the `const` keyword in C#.

Example:
csharp
```
const float Pi = 3.14159f;
```

Operators in C#

C# supports a variety of operators that allow you to perform calculations and logic on variables and values. These include:

- Arithmetic Operators: `+`, `-`, `*`, `/`, `%` (for basic mathematical operations).
- Relational Operators: `==`, `!=`, `<`, `>`, `<=`, `>=` (for comparisons).
- Logical Operators: `&&`, `||`, `!` (AND, OR, and NOT).
- Bitwise Operators: `&`, `|`, `^`, `~` (for bit-level operations).
- Assignment Operators: `=`, `+=`, `-=`, `*=`, `/=` (for assigning and updating values).
- Increment/Decrement Operators: `++`, `--` (for increasing or decreasing values by 1).

Example:
csharp
```
int a = 5;
int b = 3;
int sum = a + b; // sum = 8
```

Control Structures

Control structures in C# allow you to manage the flow of the program by making decisions and performing loops:

- if-else: Executes code based on a condition.
csharp
```
if (age >= 18) {
Console.WriteLine("You are an adult.");
} else {
Console.WriteLine("You are a minor.");
}
```

- switch-case: Evaluates an expression and matches it with a case value.
csharp
```
switch (grade) {
case 'A':
Console.WriteLine("Excellent");
break;
case 'B':
Console.WriteLine("Good");
break;
default:
Console.WriteLine("Invalid grade");
break;
}
```

- Loops: Repeatedly execute a block of code.
- `for` loop:

csharp
```
for (int i = 0; i < 5; i++) {
Console.WriteLine(i);
}
```

- `while` loop:

csharp
```
while (i < 5) {
Console.WriteLine(i);
i++;
}
```

- `do-while` loop (executes at least once):

csharp
```
do {
Console.WriteLine(i);
i++;
} while (i < 5);
```

Object-Oriented Programming in C#

C# is an object-oriented programming (OOP) language, and its core concepts include:

- Classes: Define a blueprint for objects.
- Objects: Instances of classes that hold data (fields) and behavior (methods).
- Encapsulation: Restrict access to some components of an object using access modifiers (e.g., `private`, `public`).
- Inheritance: Allow a class to inherit members of another class, promoting code reuse.
- Polymorphism: The ability of different objects to respond to the same method call in different ways.
- Abstraction: Hide complex details and expose only essential features through abstract classes or interfaces.

Example:

csharp
```
public class Animal {
public void Speak() {
Console.WriteLine("The animal makes a sound.");
}
}

public class Dog : Animal {
public void Speak() {
Console.WriteLine("The dog barks.");
}
}
```

Methods in C#

Methods are functions defined within a class that perform a specific task.

Syntax:
csharp
```
return_type MethodName(parameters) {
// Method body
return value;
}
```

Example:
csharp
```
public int AddNumbers(int a, int b) {
return a + b;
}
```

Methods can also be overloaded by defining multiple methods with the same name but different parameters.

Properties in C#

Properties provide a way to read, write, or compute the value of a private field. They are a key part of C#'s encapsulation mechanism.

Syntax:
csharp
```
public data_type PropertyName { get; set; }
```

Example:
csharp
```
private int _age;
public int Age {
get { return _age; }
set { _age = value; }
}
```

Exception Handling

C# provides a robust mechanism for handling exceptions using `try`, `catch`, and `finally` blocks.

- try: The block of code that may cause an exception.
- catch: The block of code that handles the exception.
- finally: Code that runs regardless of whether an exception occurred or not.

Example:
csharp
```
try {
int result = 10 / 0;
} catch (DivideByZeroException ex) {
Console.WriteLine("Error: " + ex.Message);
} finally {
Console.WriteLine("This code runs no matter what.");
}
```

Collections in C#

C# provides several collection types to store and manipulate groups of objects:

- Arrays: Fixed-size collection of elements of the same type.
csharp
```
int[] numbers = { 1, 2, 3, 4, 5 };
```

- Lists: A dynamic collection that can grow and shrink.
csharp
```
List list = new List { 1, 2, 3 };
list.Add(4);
```

- Dictionaries: Store key-value pairs for fast lookups.
csharp
```
Dictionary ages = new Dictionary {
{ "John", 25 },
{ "Sarah", 30 }
};
```

Delegates and Events

A delegate is a type that references a method, and it is used to pass methods as arguments.

Example:
csharp
```
public delegate void MyDelegate(string message);

public void ShowMessage(string message) {
Console.WriteLine(message);
}

MyDelegate del = new MyDelegate(ShowMessage);
del("Hello, C#!");
```

Events provide a way for a class to notify other classes or objects when something of interest occurs. Events are based on delegates.

Example:
csharp
```
public event EventHandler MyEvent;

protected virtual void OnMyEvent(EventArgs e) {
if (MyEvent != null)
MyEvent(this, e);
}
```