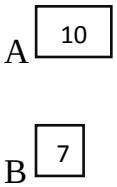
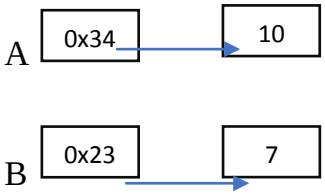
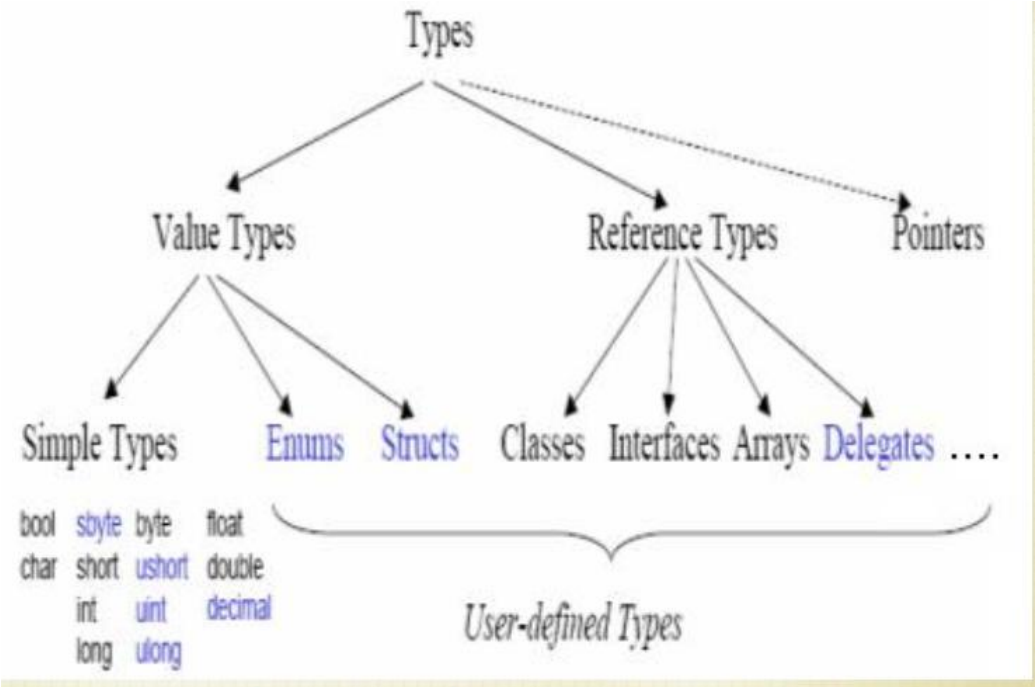
ÔN TẬP CUỐI KỲ
LẬP TRÌNH TRỰC QUAN

Câu 1: **So sánh value type và reference type.**

Value type (kiểu giá trị)	Reference type (kiểu tham chiếu)
- Gồm: + Simple Type: bool, char, int, long,.. + Enums + Struct - Lưu trực tiếp giá trị vào ô nhớ (được lưu trong stack)	- Gồm: + Class + Interface + Arrays + Delegate.. - Lưu địa chỉ và tham chiếu đến ô nhớ của giá trị (được lưu trong Heap)
Vd: + int A = 10; int B = 7; 	Vd: + CMyInt A = 10; CMyInt B = 7; 

*Câu 2:*__Những kiểu dữ liệu sau thuộc loại nào: int, float, enum, struct, class, interface,delegate, string, pointer.__



*Câu 3:*__Giải thích từ khóa ref, out, param trong C#.__

ref	out	params
Tham số đi theo sau phải khởi tạo trước khi truyềnvào phương thức	Tham số không cần khởi tạo trước khi truyền vàophương thức	Tham số nhận đối số mà số lượng đối số là biến, từ khoá này thường sử dụng tham số là mảng

*Câu 4:*__Dynamic Binding là gì? cho ví dụ minh họa.__

Dynamic Binding	Static Binding
- Kiểu của đối tượng được quyết định tại runtime	- Kiểu của đối tượng quyết định bởi Compiler (compiler time)

classCCat
{
publicvoideat(){
MessageBox.Show("eat..");
}
}
 	
void Main(string[] args)
{
CCat c = newCCat();
c.eat();
}
	abstractclassCAnimal
	{
	publicvoideat()
	{
	MessageBox.Show("eat abs");
	}
	}
	
	classCCat :CAnimal {
	publicvoideat()
	{
	MessageBox.Show("eat");
	}
	}
	void Main(string[] args)
	{
	CCat c = newCCat();
	c.eat();
	}

	Kiểu đối tượng không thể được quyết định bởi Compiler, bởi vì sự thể hiện của CCat cũng là một sự thể hiện của Animal. Vì thế Compiler không biết kiểu nào của nó, chỉ biết đến kiểu cơ sở.

*Câu 5:*__Nêu các điểm khác nhau về tính chất hướng đối tượng giữa C++ và C#.__

C++	C#
- Không phải là ngôn ngữ thuần OOP, được phát triển từ C.	- Là ngôn ngữ lập trình hướng đối tượng được phát triển bởi Microsoft.
- Hỗ trợ đa kế thừa. . .	- Chỉ có đơn kế thừa, không hỗ trợ đa kế thừa, nhưng có thể cài đặt nhiều interface trừu tượng (abstract interfaces).
- Class và struct hầu như giống nhau. . .	- Class và struct khá khác nhau. Struct không hỗ trợ kế thừa, và không hỗ trợ constructor mặc định.
- Nếu không phải là con trỏ hoặc biến tham khảo thì mọi biến được truyền bằng tham trị. .	- Đối tượng (một thực thể của class) được truyền tham khảo (truyền tham biến), struct được truyền tham trị, nếu không khai báo "ref" hoặc "out".
- C++ sử dụng cú pháp <class_name>::method(); . .	- Hỗ trợ từ khóa "base" cho việc gọi lại các phương thức của lớp cha. Hàm nạp chồng (overload) được khai báo 1 cách tường minh bằng từ khóa "override".
- Phương thức trong lớp con chắc chắn che khuất phương thức trong lớp cha nếu chúng cùng prototype.	- Phải dùng từ khóa "new" trước phương thức hay thuộc tính muốn che phủ. .
- Hàm main của C++ không thuộc về lớp nào. Là một hàm cục bộ.	- Hàm Main (M viết hoa) luôn khai báo là static.

* **Ghi chú:***dưới đây là sự khác nhau giữa C++ và C# (không chỉ về tính OOP), đọc thêm cho biết.*

- Thừa kế: Trong C++, class và struct hầu như giống nhau, ngược lại trong C#, chúng khá khác nhau. Lớp C# có thể hiện thực/thừa kế (implement) nhiều interface, nhưng chỉ được thừa kế từ chỉ 1 lớp cơ sở. Hơn nữa, struct C# không hỗ trợ thừa kế, và không hỗ trợ constructor mặc định.

- Mảng (Array): Trong C++ mảng chính là 1 con trỏ (pointer). Trong C#, mảng là đối tượng (object),có thuộc tính và phương thức. Ví dụ, số phần tử của mảng có thể lấy thông qua thuộc tính Length. Mảng C# còn kiểm tra chỉ số thứ tự khi truy xuất phần tử. Cú pháp khai báo mảng trong C# và C++ cũng khác: token "[] " theo sau khai báo kiểu của mảng chứ không phải đứng sau tên biến như trong C++.

- Boolean: Trong C++, kiểu bool thực ra là số integer. Trong C#, không có sự tự chuyển đổi qua lại giữa bool và các kiểu khác.

- Kiểu số nguyên long: Trong C#, kiểu long có độ dài 64 bit, trong C++ là 32 bit.

- Kiểu số nguyên int: Trong C#, nó chắc chắn có độ dài 32 bit, trong VC++ là 32 bit, trong các trình biên dịch C++ khác, độ dài của nó có thể là 16 bit.

- Truyền tham số: Trong C++, mọi biến được truyền bằng trị, nếu không phải là con trỏ hoặc biến tham khảo. Trong C#, đối tượng (một thực thể của class) được truyền tham khảo (truyền tham biến), struct được truyền tham trị, nếu không khai báo "ref" hoặc "out".

- Phát biểu switch: Không giống C++, C# hỗ trợ việc nhảy từ nhãn này qua nhãn khác qua phát biểu goto.

- Delegate: Delegate C# giống như con trỏ hàm của C++, nhưng an toàn và bảo mật.

- Phương thức trong lớp cha: C# hỗ trợ từ khóa "base" cho việc gọi lại các phương thức của lớp cha. Hàm nạp chồng (overload) được khai báo 1 cách tường minh bằng từ khóa "override".

- Che khuất phương thức: Trong C++ phương thức trong lớp con chắc chắn che khuất phương thức trong lớp cha nếu chúng cùng prototype. Trong C#, bạn phải dùng từ khóa "new" trước phương thức hay thuộc tính bạn muốn che phủ.

- Chỉ thị tiền biên dịch: C# chỉ cho đặt dẫn hướng tiền biên dịch (là các symbol) trong configuration của project, mà không có các file header (.h).

- Exception: C# cung cấp từ khóa "finally" cho đoạn code try {} catch để bắt Exception mà cần thêm những tác vụ bảo đảm an toàn.

- Toán tử: C# hỗ trợ một số toán tử mở rộng như: "is", "typeof".

- Từ khóa "extern": Trong C++, "extern" dùng để khai báo kiểu/biến được định nghĩa trong file .obj khác. Trong C#, extern dùng để tạo ra một bí danh (alias) khi sử dụng version khác của 1 assembly.

- Từ khóa "static": Trong C++, static dùng trong khai báo phương thức/thuộc tính/biến trong class, hàm. Trong C#, chỉ dùng đối với các thành viên của class, không có trong hàm.

- Hàm main: Trong C# hàm Main (M viết hoa) luôn khai báo là static.

- Con trỏ: Được phép trong C#, nhưng chỉ trong mode unsafe.

- Chuỗi (string): Trong C++, string chỉ đơn giản là mảng ký tự. Trong C# string là 1 đối tượng (object) có hỗ trợ các phương thức tìm kiếm.

- Từ khóa foreach: Trong C#, cho phép duyệt qua các phần tử của mảng hoặc tập hợp.

- Tầm vực Toàn cục: Trong C#, biến phương thức không có tầm vực toàn cục, chúng bắt buộc phải thuộc 1 class/struct nào đó.

- Biến cục bộ: Trong C#, biến cục bộ có thể dùng trước khi khởi tạo giá trị.

- Quản lý bộ nhớ: C++ không có cơ chế dọn rác, bộ nhớ không được trả lại hệ thống cho đến khi process kết thúc hoặc tự giải phóng bằng phát biểu delete/free. C# làm một cách tự động.

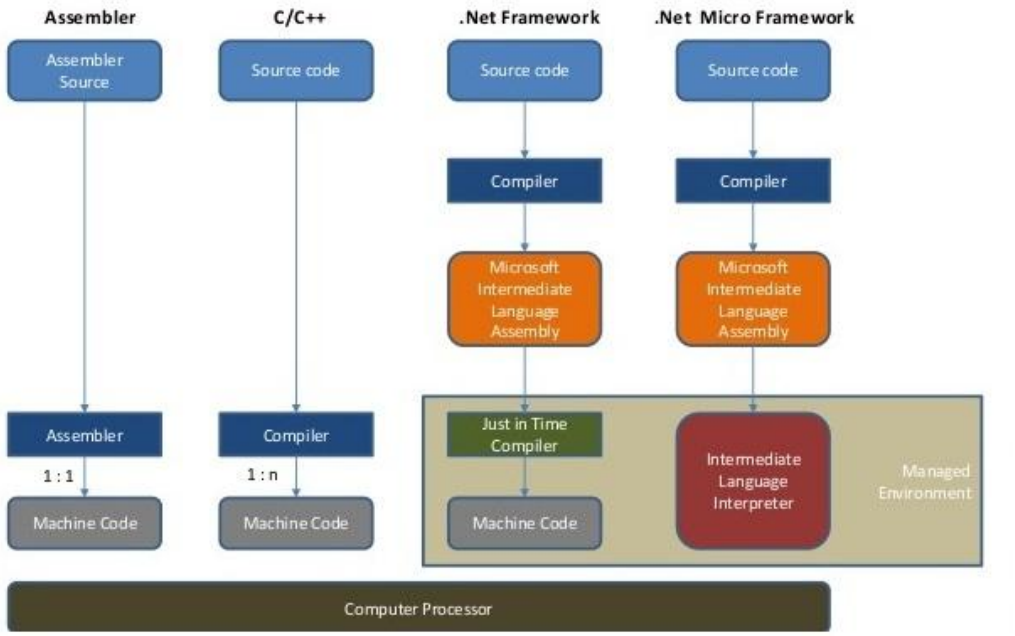
- Hàm khởi tạo: Tương tự như C++, nếu bạn không viết hàm khởi tạo nào trong 1 lớp của C# thì 1 hàm khởi tạo mặc định tự động tạo ra (trong assembly) cho bạn. Hàm khởi tạo mặc định khởi tạo những giá trị mặc định cho các biến thành viên.

- Giá trị mặc định cho tham số hàm/phương thức: C# không có, bạn phải dùng cách nạp chồng phương thức để thực hiện.

- Kiểu generic: Khai báo như là một tham số của class hay method, tương tự như template trong C++.

- Từ khóa "as": toán tử chuyển đổi kiểu, sẽ ném một exception nếu việc chuyển đổi không thực hiện được. Giống với static_cast trong C++, nhưng khác với dynamic_cast vì dynamic_cast không thực hiện kiểm tra sự tương thích kiểu nên không ném exception nào nếu như thực sự có lỗi.

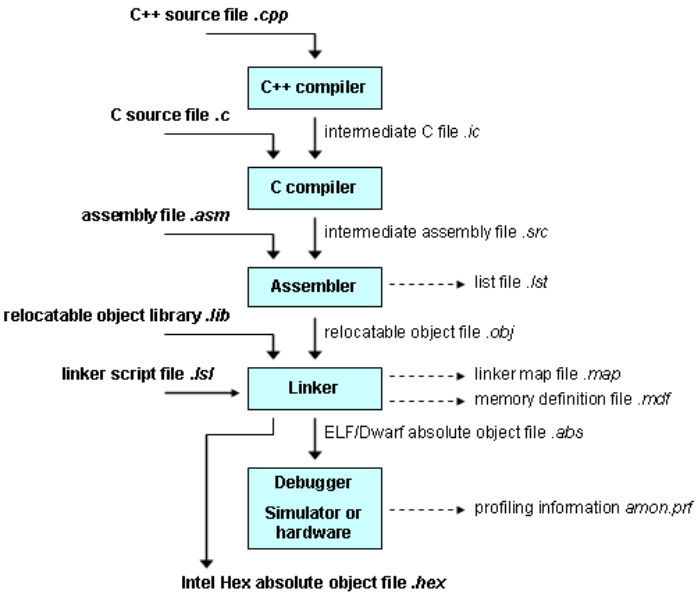
*Câu 6:*__So sánh tốc độ thực thi giữa C# và C++.__

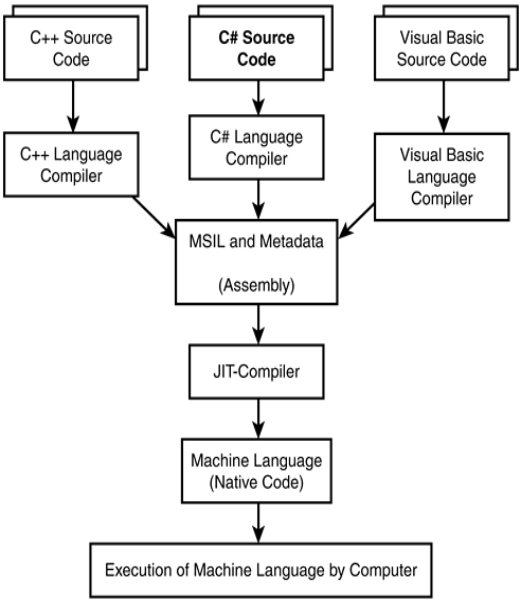


Nhìn hình >> là có thể thấy ngay rồi, (tự chém đê.)

+ C/C++ được compiler biên dịch từ: C/C++ -> (mã máy) Machine Code (biên dịch 1 lần)

+ C# được compiler biên dịch từ : C# -> CIL code (MSIL) -> (mã máy) Machine Code (biên dịch 2 lần)





*Câu 7:*__Phân biệt Abstract Class vs Interface (Không biết 2 thứ này có liên quan nhau luôn).__

Abstract class là gì?

- Abstract class hay còn được gọi là lớp trừu tượng.
- Abstract class được xem như một class cha cho tất cả các class con có cùng bản chất, và các lớp con(lớp dẫn xuất) sẽ chỉ kế thừa từ một lớp trừu trượng.
- Lớp trừu tượng này không cho phép khởi tạo các tham số, chỉ được phép khai báo hay còn gọi là abstract không có cài đặt bên trong.
- Trong các lớp Abstract không cho phép khơi tạo đối tượng(instance) của lớp ấy, có nghĩa không cho phép khởi tạo giá trị.

Interface là gì?

- Interface hay còn gọi là lớp giao diện.

- Lớp Interface được xem nư một mặt nạ cho các class cùng các thức hoạt động nhưng có thể cùng hoặc khác nhau về bản chất bên trong.
- Các lớp dẫn xuất(lớp con) có thể kế thừa từ nhiều lớp interface khác nhau để bổ sung đầy đủ cách thức hoạt động cho nó. Hay còn được gọi là Multiple inheritance(đa kế thừa).

Khác nhau

Abstract class	Interfaces
- Có thể kế thừa một class và nhiều interface	- Chỉ có thể kế thừa nhiều interface.
- Một abstract class có thể có nhiều phương thức có thân bên trong.	- Không có.
- Các phương thức của abstract class được thực thi sử dụng từ khóa override.	- Không cần.
- Là lựa chọn thích hợp khi vừa khai báo các phương thức thông thường vừa khai báo các phương thức abstract.	- Thích hợp cho việc khai báo duy nhất các phương thức abstract.
- Có thể có hàm khởi tạo và destructor	- Không có.

(Bổ sung) -Không hỗ trợ đa thừa kế -Có

-Các phương thức có thể xác định modifier -Mọi phương thức, property đều có

tầm vực public

-Có các fields và constants -Không có

*Câu 8:*__Sự khác nhau giữa class, object, struct và interface?__

Một class là một tập hợp các đối tượng(objects) và mô tả của các đối tượng có chung các thuộc tính và hành động. Nó chứa các đặc tính của đối tượng như các thuộc tính(attributes), các hành động hoặc các hành vi (behaviors).

Trong C#, một cấu trúc (structure) là một kiểu dữ liệu. Nó giúp bạn tạo một biến đơn mà giữ dữ liệu liên quan của các kiểu dữ liệu đa dạng. Từ khóa struct trong C# được sử dụng để tạo một cấu trúc (structure). Các cấu trúc được sử dụng để biểu diễn một bản ghi (record). Các cấu trúc trong C# là khá khác với kiểu cấu trúc truyền thống trong C hoặc C++. Cấu trúc trong C# có các đặc điểm sau:

- Cấu trúc có thể có các phương thức, các trường, indexer, thuộc tính, phương thức operator, và sự kiện.
- Cấu trúc có thể có các constructor đã được định nghĩa, nhưng không có destructor. Tuy nhiên, bạn không thể định nghĩa một constructor mặc định cho một cấu trúc. Constructor mặc định được định nghĩa tự động và không thể bị thay đổi.
- Không giống các Lớp, cấu trúc không thể kế thừa từ cấu trúc hoặc lớp khác.
- Cấu trúc không thể được sử dụng như là một cơ sở cho cấu trúc hoặc lớp khác.
- Một cấu trúc có thể triển khai một hoặc nhiều Interface.
- Thành viên cấu trúc không thể được xác định ở dạng abstract, virtual, hoặc protected.
- Khi bạn tạo một đối tượng Struct bởi sử dụng toán tử new, nó lấy đối tượng đã tạo và constructor thích hợp được gọi. Không giống Lớp, cấu trúc có thể được khởi tạo mà không cần sử dụng toán tử new.
- Nếu toán tử new không được sử dụng, thì các trường chưa được gán và đối tượng không thể được sử dụng tới khi tất cả trường đó được khởi tạo.

Interface định nghĩa các thuộc tính, phương thức và sự kiện, mà là các thành viên của Interface đó. Các Interface chỉ chứa khai báo của các thành viên này. Việc định nghĩa các thành viên là trách nhiệm của lớp kế thừa. Nó thường giúp ích trong việc cung cấp một Cấu trúc chuẩn mà các lớp kế thừa nên theo. Các Interface được khai báo bởi sử dụng từ khóa interface trong C#. Nó tương tự như khai báo lớp. Theo mặc định, các lệnh Interface là puclic.

Một class định nghĩa một kiểu của đối tượng. Một đối tượng là một thực thể cụ thể trên cơ sở của một class, và nó đôi khi còn được gọi là một thể hiện của một class.

*Câu 9:***Namespace là gì và cho ví dụ 3 namespace mà bạn biết và mục đích sử dụng của từng namespace.**

Namespace(Không gian tên) trong C# là kỹ thuật phân hoạch không gian các định danh, các kiểu dữ liệu thành những vùng dễ quản lý hơn, nhằm tránh sự xung đột giữa việc sử dụng các thư viện khác nhau từ các nhà cung cấp.Các tên lớp được khai báo trong một namespace sẽ không xung đột với cùng tên đó của lớp được khai báo trong namespace khác.

Ví dụ:

Namespace System.Drawing: The System.Drawing namespace provides access to GDI+ basic graphics functionality (tạm dịch: namespace cung cấp khả năng truy cập các chức năng đồ họa cơ bản GDI+)

Namespace Microsoft.CSharp: The Microsoft.CSharp namespace contains classes that support compilation and code generation using the C# language. (Tạm dịch: namespace có chứa các class hỗ trợ trong việc biên dịch và tạo mã bằng ngôn ngữ C#)

Namespace System.Linq: The System.Linq namespace provides classes and interfaces that support queries that use Language-Integrated Query (LINQ).(Tạm dịch: Namespace cung cấp các lớp và giao diện hỗ trợ các truy vấn dùng LINQ)

Câu 10: .Trình bày hiểu biết của em về CLR và .Net FrameWork

.NET Framework của Microsoft là một nền tảng lập trình tập hợp các thư viện lập trình có thể được cài thêm hoặc đã có sẵn trong các hệ điều hành Windows. Nó cung cấp những giải pháp thiết yếu cho những yêu cầu thông thường của các chương trình điện toán như lập trình giao diện người dùng, truy cập dữ liệu, kết nối cơ sở dữ liệu, ứng dụng web, các giải thuật số học và giao tiếp mạng. Ngoài ra, .NET Framework quản lý việc thực thi các chương trình được viết dựa trên .NET Framework do đó người dùng cần phải cài .NET Framework để có thể chạy các chương trình được viết trên nền .NET.

Framework có 2 thành phần chính là:

1. Common Language Runtime (CLR)bộ phận quản lí đoạn mã .NET khi nó chạy.

Dưới đây là các bộ phận của CLR và nhiệm vụ của từng bộ phận:

- Garbage Collection (GC): Bộ phận thu gom rác.

CLR tự động quản lí kí ức, nhằm làm giảm sự rò rỉ kí ức. Một lúc nào đó (do CLR tự chọn), GC sẽ nhảy vào giải phóng các vùng kí ức không còn được cái gì trỏ đến (không dùng nữa); lập trình viên không phải làm việc này (trừ lúc cố tình).

- Code Access Security: (CAS)

CAS cung cấp quyền hạn cho các chương trình, tùy thuộc vào các thiết lập bảo mật của máy. Chẳng hạn, thiết lập bảo mật của máy cho phép chương trình chạy trên đó được sửa hay tạo file mới, nhưng không cho phép nó xóa file. CAS sẽ chăm sóc các đoạn mã, không cho phép chúng làm trái với các qui định này.

- Code Verification: Bộ phận chứng nhận đoạn mã.

Nó đảm bảo cho việc chạy các đoạn mã là đúng đắn, không loạng quạng, và đảm bảo an toàn kiểu dữ liệu. Nó ngăn chặn các đoạn mã hành động phi pháp như truy nhập vào các vùng kí ức không được phép.

- IL-to-native translators and optimizer's: Bộ phận chuyển IL thành mã máy và tối ưu chương trình.

CLR dùng trình biên dịch JIT để chuyển các đoạn mã IL thành mã máy và chạy chúng. CLR đồng thời dựa vào đặc điểm của máy mà tinh chỉnh để tối ưu việc thực thi đoạn mã đó.

2. NET Framework class library

*Câu 11:***Phân biệt giữa lập trình trên môi trường DOS và Windows.**

DOS	
- Thực hiện tuần tự. .	- Multi-tasking. - Multi-CPU.
- Single Task. - Single CPU.	- Tích hợp sẵn Multimedia. . - Hỗ trợ 32 bits hoặc hơn nữa.
- Phải dùng các thư viện Multimedia riêng. - Ứng dụng 16 bits.	- Không hỗ trợ nhiều công nghệ.
Windows	
- Lập trình sự kiện dựa vào thông điệp (message).	- Hỗ trợ nhiều công nghệ DDL, COM, DDE, OLE…

*Câu 12:*__Mô tả các sự kiện trong vòng đời của form.__

Vòng đời của form thể hiện qua dãy các sự kiện như sau:

- Move:

+ Một form bắt đầu khởi động khi phương thức khởi tạo được thực hiện, nó gọi phương thức InitializeComponent để khởi tạo các đối tượng con. InitializeComponent được trình sinh mã tạo ra, vì vậy không nên thêm mã vào trong phương thức này. Nếu muốn thêm các điều khiển khác, có thể thực hiện trong phương thức khởi tạo sau khi gọi phương thức InitializeComponent.

+ Khi gọi phương thức Show hoặc ShowDialog, form mới và các điều khiển con được hiển thị. Chú ý khi hiện form lên màn hình thì sự kiện Load sẽ được kích hoạt.

- Load: Sự kiện load rất hữu ích khi muốn khởi tạo sau cùng trước khi form hiện lên.

- Visible changed: Khi form được nạp lần đầu, nó trở thành form hoạt động, nó được hiện lên trên và được nhận phím. Đây là thời điểm sự kiện Activated kích hoạt.

- Activated (Deactivation và Reactivation): Khi người dùng quay lại ứng dụng, sự kiện Activated lại được kích hoạt, cho phép tiếp tục các hoạt động mà bạn đã dừng khi form ở trạng thái không hoạt động.

- Shown: Có thể cho hiện hoặc ẩn form thông qua thuộc tính Visible hoặc là phương thức Show, Hide.

- Closing: Khi không dùng ứng dụng nữa, người dùng có thể đóng theo các cách như File/Close, Alt+F4, hoặc nút đóng. Khi đó phương thức Close của form được gọi. Trước khi đóng form thì sự kiện Closing được kích hoạt, lúc này form chưa bị đóng. Ví dụ hỏi người dùng khi đóng form, nếu chọn No thì không đóng form. Sự kiện Closing là nơi tốt nhất để thực hiện lưu các thuộc tính cần thiết của form khi mở lại, vd như kích thước, vị trí form.

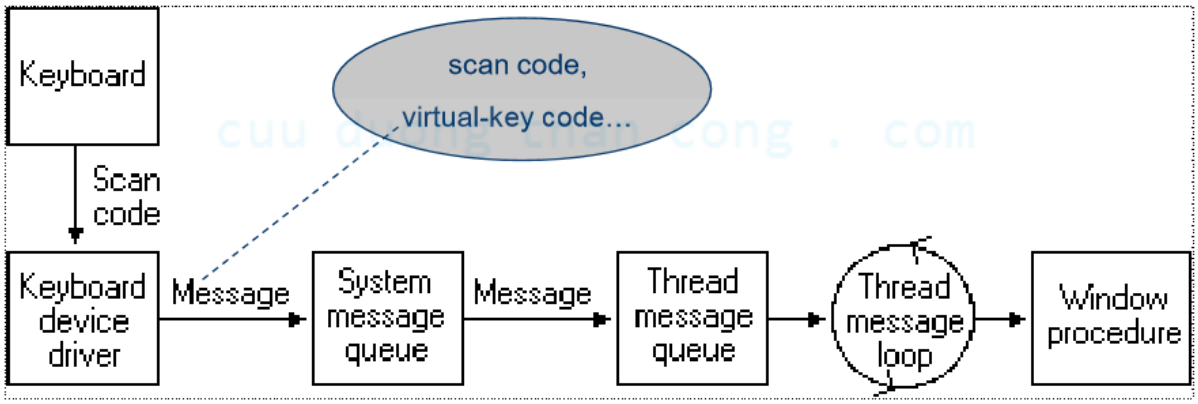
- Closed: Tắt giao diện form, giải phóng tài nguyên, lưu thông tin, cập nhật form cha.

*Câu 13:***Sự khác nhau giữa delegate và event?**

– Event có thể được khai báo trong interface, Delegate thì không.

– Event chỉ có thể được gọi (invoked) ở bên trong class chứa nó, Delegate thì có thể được gọi ở bất cứ đâu (Tùy thuộc vào access modifier).

*Câu 14:***Mô hình xử lý sự kiện bàn phím của Windows.**



*Câu 15:***Trình bày các phương thức xử lý sự kiện chuột trên form.**

- Để xử lý sự kiện MouseDown ta override phương thức OnMouseDown. Ví dụ:

```
protected override void OnMouseDown(MouseEventArgs mea)
{
    MessageBox.Show("Ban vua nhan chuot " + mea.Button);
}
```

- Để xử lý sự kiện MouseUp ta override phương thức OnMouseUp. Ví dụ

```
protected override void OnMouseUp(MouseEventArgs mea)
{
    MessageBox.Show("Ban vua nha chuot " + mea.Button);
}
```

-Để xử lý sự kiện MouseMove ta override phương thức OnMouseMove. Ví dụ:

```
protected override void OnMouseMove(MouseEventArgs mea)
{
            //Ve mot duong thang tu toa do (0,0) den toa do chuot di chuyen
            Graphics g = CreateGraphics();
            Pen pen = new Pen(System.Drawing.Color.Blue);
            g.DrawLine(pen, 0, 0, mea.X, mea.Y);
}
```

- Để xử lý sự kiện MouseWheel e ta override phương thức OnMouseWheel. Ví dụ:

```
protected override void OnMouseWheel(MouseEventArgs mea)
{
        if (mea.Delta>0)
            MessageBox.Show("Ban vua scroll chuot len", "Thong bao");
        else
            MessageBox.Show("Ban vua scroll chuot xuong", "Thong bao");
}
```