**Tổng quan về ngôn ngữ C#:**

*Câu 1: Sự khác nhau giữa ngôn ngữ lập trình dựa trên thành phần (Component-Based) và hướng đối tượng (Object- Oriented)?*

Lập trình hướng dựa trên thành phần (COP) là kiểu lập trình có xu hướng chia hệ thống phần mềm thành những thành phần giữ các chức năng khác nhau (mỗi thành phần này được gọi là một bộ phận hợp thành) mà khi kết hợp lại ta thu được một hệ thống phần mềm hoàn chỉnh.

Lập trình hướng đối tượng (gọi tắt là OOP là kĩ thuật lập trình hỗ trợ công nghệ đối tượng. OOP được xem là giúp tăng năng suất, đơn giản hóa độ phức tạp khi bảo trì cũng như mở rộng phần mềm bằng cách cho phép [lập trình viên](http://vi.wikipedia.org/wiki/L%E1%BA%ADp_tr%C3%ACnh_vi%C3%AAn) tập trung vào các đối tượng phần mềm ở bậc cao hơn.

Sự khác nhau giữa 2 phương pháp lập trình OOP và COP được chỉ ra trong bảng sau:

|  |  |
| --- | --- |
| **COP** | **OOP** |
| COP phát triển phần mềm bằng cách lắp ráp các thành phần | OOP nhấn mạnh cài đặt các lớp và các đối tượng |
| COP nhấn mạnh giao diện và kết cấu | OOP nhấn mạnh về đối tượng/lớp  OOP hỗ trợ bao bọc, thừa kế, đa xạ nhưng chưa bao giờ đạt mục đích của nó vì kế thừa xâm phạm bao bọc, hơn nữa các đối tượng và các lớp không tự vận hành. |
| COP là kỹ thuật đóng gói và phân tán | OOP là kỹ thuật cài đặt |
| COP hỗ trợ sử dụng lại mức cao | OOP hỗ trợ sử dụng lại mức thấp |
| COP về nguyên tắc có thể viết bởi bất kỳ ngôn ngữ lập trình nào | OOP bị giới hạn bởi ngôn ngữ OO |
| COP gồm các thành phần gắn kết lỏng lẻo | OOP gồm các đối tượng phụ thuộc chặt chẽ hơn vào các đối tượng khác qua giao diện kế thừa (Cha thay đổi, con thay đổi theo) |
| COP có các thành phần hạt nhân từ lớn đến nhỏ | OOP gồm các đối tượng là các đơn vị của kiến trúc |
| COP hỗ trợ đa giao diện và thiết kế hướng giao diện | OOP không cung cấp mỗi quan hệ rõ ràng về giao diện giữa các lớp con và lớp cha |

*Câu 2 : Những đặc điểm chính của ngôn ngữ lập trình hướng đối tượng? Mô tả sơ lược các đặc điểm đó?*

Lập trình hướng đối tượng là một phương pháp lập trình có các tính chất chính sau:

* *Tính trừu tượng (abstraction):* Đây là khả năng của chương trình bỏ qua hay không chú ý đến một số khía cạnh của thông tin mà nó đang trực tiếp làm việc lên, nghĩa là nó có khả năng tập trung vào những cốt lõi cần thiết. Mỗi đối tượng phục vụ như là một "động tử" có thể hoàn tất các công việc một cách nội bộ, báo cáo, thay đổi trạng thái của nó và liên lạc với các đối tượng khác mà không cần cho biết làm cách nào đối tượng tiến hành được các thao tác. Tính chất này thường được gọi là sự trừu tượng của dữ liệu.  
  Tính trừu tượng còn thể hiện qua việc một đối tượng ban đầu có thể có một số đặc điểm chung cho nhiều đối tượng khác như là sự mở rộng của nó nhưng bản thân đối tượng ban đầu này có thể không có các biện pháp thi hành. Tính trừu tượng này thường được xác định trong khái niệm gọi là lớp trừu tượng hay lớp cơ sở trừu tượng.
* *Tính đóng gói (encapsulation) và che giấu thông tin (information hiding):* Tính chất này không cho phép người sử dụng các đối tượng thay đổi trạng thái nội tại của một đối tượng. Chỉ có các phương thức nội tại của đối tượng cho phép thay đổi trạng thái của nó. Việc cho phép môi trường bên ngoài tác động lên các dữ liệu nội tại của một đối tượng theo cách nào là hoàn toàn tùy thuộc vào người viết mã. Đây là tính chất đảm bảo sự toàn vẹn của đối tượng.
* *Tính đa hình (polymorphism):* Thể hiện thông qua việc gửi các thông điệp (message). Việc gửi các thông điệp này có thể so sánh như việc gọi các hàm bên trong của một đối tượng. Các phương thức dùng trả lời cho một thông điệp sẽ tùy theo đối tượng mà thông điệp đó được gửi tới sẽ có phản ứng khác nhau. Người lập trình có thể định nghĩa một đặc tính (chẳng hạn thông qua tên của các phương thức) cho một loạt các đối tượng gần nhau nhưng khi thi hành thì dùng cùng một tên gọi mà sự thi hành của mỗi đối tượng sẽ tự động xảy ra tương ứng theo đặc tính của từng đối tượng mà không bị nhầm lẫn.
* *Tính kế thừa (inheritance):* Đặc tính này cho phép một đối tượng có thể có sẵn các đặc tính mà đối tượng khác đã có thông qua kế thừa. Điều này cho phép các đối tượng chia sẻ hay mở rộng các đặc tính sẵn có mà không phải tiến hành định nghĩa lại. Tuy nhiên, không phải ngôn ngữ định hướng đối tượng nào cũng có tính chất này.

*Câu 3 : C# chia làm mấy kiểu dữ liệu chính? Nếu ta tạo một lớp tên myClass thì lớp này được xếp vào kiểu dữ liệu nào?*

C# là ngôn ngữ lập trình mạnh về kiểu dữ liệu, một ngôn ngữ mạnh về kiểu dữ liệu là phải khai báo kiểu của mỗi đối tượng khi tạo (kiểu số nguyên, số thực, kiểu chuỗi, kiểu điều khiển...) và trình biên dịch sẽ giúp cho người lập trình không bị lỗi khi chỉ cho phép một loại kiểu dữ liệu có thể được gán cho các kiểu dữ liệu khác. Kiểu dữ liệu của một đối tượng là một tín hiệu để trình biên dịch nhận biết kích thước của một đối tượng (kiểu int có kích thước là 4 byte) và khả năng của nó (như một đối tượng button có thể vẽ, phản ứng khi nhấn,...).

C# chia thành hai tập hợp kiểu dữ liệu chính: Kiểu xây dựng sẵn (built- in) mà ngôn ngữ cung cấp cho người lập trình và kiểu được người dùng định nghĩa (user-defined) do người lập trình tạo ra. C# phân tập hợp kiểu dữ liệu này thành hai loại: *Kiểu dữ liệu giá trị* (value) và *kiểu dữ liệu tham chiếu* (reference). Việc phân chi này do sự khác nhau khi lưu kiểu dữ liệu giá trị và kiểu dữ liệu tham chiếu trong bộ nhớ. Đối với một kiểu dữ liệu giá trị thì sẽ được lưu giữ kích thước thật trong bộ nhớ đã cấp phát là stack. Trong khi đó thì địa chỉ của kiểu dữ liệu tham chiếu thì được lưu trong stack nhưng đối tượng thật sự thì lưu trong bộ nhớ heap. Nếu chúng ta có một đối tượng có kích thước rất lớn thì việc lưu giữ chúng trên bộ nhớ heap rất có ích.

Tất cả các kiểu dữ liệu xây dựng sẵn là kiểu dữ liệu giá trị ngoại trừ các đối tượng và *chuỗi*. Và tất cả các kiểu do người dùng định nghĩa ngoại trừ kiểu cấu trúc đều là kiểu dữ liệu tham chiếu. *Do đó nếu ta tạo một lớp tên myClass thì lớp này được xếp vào kiểu dữ liệu tham chiếu.*

*Ngoài ra C# cũng hỗ trợ một kiểu con trỏ C++, nhưng hiếm khi được sử dụng, và chỉ khi nào làm việc với những đoạn mã lệnh không được quản lý (unmanaged code). Mã lệnh không được quản lý là các lệnh được viết bên ngoài nền .MS.NET, như là các đối tượng COM.*

*Câu 4 : Kiểu chuỗi trong C# là kiểu dữ liệu nào?*

Xem lại câu 3. Tất cả các kiểu dữ liệu xây dựng sẵn là kiểu dữ liệu giá trị ngoại trừ các đối tượng và *chuỗi*. 🡪 kiểu chuỗi trong C# là kiểu dữ liệu dữ liệu tham chiếu

*Câu 5 :* *Dữ liệu của biến kiểu dữ liệu tham chiếu được lưu ở đâu trong bộ nhớ?*

Xem lại câu 3: Đối với một kiểu dữ liệu giá trị thì sẽ được lưu giữ kích thước thật trong bộ nhớ đã cấp phát là stack. Trong khi đó thì địa chỉ của kiểu dữ liệu tham chiếu thì được lưu trong stack nhưng đối tượng thật sự thì lưu trong bộ nhớ heap.

*Câu 6 : Sự khác nhau giữa chuyển đổi tường minh và chuyển đổi ngầm định?*

Những đối tượng của một kiểu dữ liệu này có thể được chuyển sang những đối tượng của một kiểu dữ liệu khác thông qua cơ chế chuyển đổi tường minh hay ngầm định. Chuyển đổi nhầm định được thực hiện một cách tự động, trình biên dịch sẽ thực hiện công việc này. Còn chuyển đổi tường minh diễn ra khi chúng ta gán ép một giá trị cho kiểu dữ liệu khác. Việc chuyển đổi giá trị ngầm định được thực hiện một cách tự động và đảm bảo là không mất thông tin. Ví dụ, chúng ta có thể gán ngầm định một số kiểu short (2 byte) vào một số kiểu int (4 byte) một cách ngầm định. Sau khi gán hoàn toàn không mất dữ liệu vì bất cứ giá trị nào của short cũng thuộc về int:

short x = 10;  
int y = x; // chuyển đổi ngầm định

Tuy nhiên, nếu chúng ta thực hiện chuyển đổi ngược lại, chắc chắn chúng ta sẽ bị mất thông tin. Nếu giá trị của số nguyên đó lớn hơn 32.767 thì nó sẽ bị cắt khi chuyển đổi. Trình biên dịch sẽ không thực hiện việc chuyển đổi ngầm định từ số kiểu int sang số kiểu short:

short x;  
int y = 100;  
x = y; // Không biên dịch, lỗi !!! Để không bị lỗi chúng ta phải dùng lệnh gán tường minh, đoạn mã trên được viết lại như sau:  
short x;  
int y = 500;  
x = (short) y; // Ép kiểu tường minh, trình biên dịch không báo lỗi

*Câu 7 : Tại sao ta dùng struct thay vì class cho vector và matrix?*

Trong C#, ta có thể xây dựng vector,matrix theo kiểu struct hoặc class. Chọn lựa kiểu dữ liệu tùy thuộc vào yêu cầu mô tả dữ liệu. Ta có so sánh giữa 2 kiểu dữ liệu này như sau:

|  |  |
| --- | --- |
| Struct | Class |
| - Là kiểu dữ liệu trực tiếp, hay struct trực tiếp chứa dữ liệu.  - Sử dụng struct sẽ giảm tải cho garbage collector  - Được lưu trong stack  - Truyền tham số kiểu struct nhanh hơn kiểu class  - Thích hợp với cấu trúc dữ liệu nhỏ, sử dụng nhiều kiểu dữ liệu dựng sẵn | - Là kiểu đối tượng, tức là chỉ chứa tham chiếu đến đối tượng  - Luôn được hủy nhờ garbage collector  - Được lưu trong heap  - Thích hợp với các cấu trúc dữ liệu phức tạp và lập trình phức tạp |

Do đó cấu trúc thường dùng để thể hiển các đối tượng không đòi hỏi một ngữ nghĩa tham chiếu, hay một lớp nhỏ mà khi đặt vào trong stack thì có lợi hơn là đặt trong bộ nhớ heap. Vì vậy ta dùng struct thay vì class cho vector và matrix

**Lớp và đối tượng**

*Câu 8 : Sự khác nhau giữa một lớp và một đối tượng của lớp?*

Lớp là một khái niệm mô tả cho những thực thể có chung tính chất và hành vi có thể nói lớp là một khuôn mẫu cho các đối tượng. Còn đối tượng là những đại diện cho lớp, mọi đối tượng đều có chung tính chất và hành vi mà lớp định nghĩa. Lớp không tạo được đối tượng: lớp trừu tượng

*Câu 9 : Tại sao phải khai báo static cho phương thức Main của lớp?*

Hàm Main được thực hiện đầu tiên trong chương trình C# nên nếu không khai báo static cho hàm Main thì phải tạo ra một lớp chứa hàm main. Mà contructor có thể được overload bất cứ đâu trong chương trình nên không có một cách chắc chắn để C# tìm thấy hàm Main này. Vì vậy, hàm Main luôn được khai báo static

*Câu 10 : C# chia làm mấy kiểu dữ liệu chính? Nếu ta tạo một lớp tên myClass thì lớp này được xếp vào kiểu dữ liệu nào?*

*Tương tự câu 3*

*Câu 11 : Một lớp có thể bao gồm những thành viên nào? Ý nghĩa của chúng?*

Các thành phần của lớp gồm: **Fields**, **Properties** và **Methods** và các thành phần này được phân làm 2 loại là static và instance.  
  
***Trong đó :***   
Fields là các phần tử dùng để thể hiện các biến trong lớp.   
Properties là phần tử dùng để truy cập đến đặc điểm của một đối tượng hoặc một class. Properties được định nghĩa bằng 2 phần, phần thứ nhất giống như định nghĩa Fields, phần thứ 2 có thêm 2 phần tử get và set.   
Methods hay phương thức chính là các “hành vi” được định nghĩa trong class. Nó dùng để thực hiện một công việc nào đó của một đối tượng hay một class. Khi học về method bạn cần đặc biệt lưu ý đến các vấn đề về Methods overloading, Constructor, Destructor và cách truyền tham số cho method.   
Lớp Abstract class và Sealed class   
Là 2 lớp đặc biệt của C# trong đó:   
Abstract class là lớp chứa phương thức Abstract hay phương thức ảo- tức là các phương thức chỉ được khai báo chứ không thực thi hành động nào. Abstract class chỉ được dùng làm lớp cha cho các lớp kế thừa.   
Sealed class là lớp không bao giờ được kế thừa. Nếu bạn khai báo một lớp dẫn xuất từ một lớp Sealed class thì chương trình sẽ báo lỗi.

*Câu 12 : Sự khác nhau giữa tham số (parameter) và đối mục (argument)?*

Tham số được định nghĩa là những thứ được truyền vào trong một phương thức.  
Một tham số xuất hiện với định nghĩa của phương thức ở đầu phương thức. Một đối mục là  
giá trị được truyền vào phương thức. Chúng ta truyền những đối mục vào phương thức phù  
hợp với những tham số đã khai báo của phương thức.

*Câu 13 : Chúng ta có thể tạo phương thức bên ngoài của lớp hay không?*

Mặc dù trong những ngôn ngữ khác, chúng ta có thể tạo các phương thức bên ngoài của lớp. Nhưng trong C# thì không, C# là hướng đối tượng, do vậy tất cả các mã nguồn phải được đặt bên trong một lớp.

*Câu 14 : Sự khác nhau giữa thành viên được khai báo là public và các thành viên không được khai báo là public?*

Khai báo public , protected , private là để giới hạn mức độ truy cập đến các thuộc tính,phương thức của một class nhằm tăng tính bảo mật cho dữ liệu. Như vậy có thể ngăn chặn sự truy xuất dữ liệu trái phép từ các class khác nhau khi chưa được phép. Như vậy tính độc lập trong chương trình càng được nâng cao. Các từ khóa thường được dùng để thể hiện đó là:  
- public: cho phép truy cập từ ngoài vào  
- protected: cho phép truy cập chỉ với lớp con  
- private : không cho phép truy cập từ ngoài

*Câu 15 : Từ khoá nào được sử dụng trong việc thực thi thuộc tính của lớp?*

Từ khóa **this** được dùng để tham chiếu đến thể hiện hiện hành của một đối tượng. Tham chiếu **this** này được xem là con trỏ ẩn đến tất các phương thức không có thuộc tính tĩnh trong một lớp. Mỗi phương thức có thể tham chiếu đến những phương thức khác và các biến thành viên thông qua tham chiếu **this** này.

Tham chiếu **this** này được sử dụng thường xuyên theo ba cách:

* Sử dụng khi các biến thành viên bị che lấp bởi tham số đưa vào
* Sử dụng tham chiếu **this** để truyền đối tượng hiện hành vào một tham số của một phương thức của đối tượng khác.

Sử dụng tham chiếu **this** là *mảng chỉ mục* (indexer)

*Câu 16 : Những kiểu dữ liệu nào được trả về từ phương thức?*

Tất cả các kiểu dữ liệu trong C có thể được trả về bởi method và method có thể không trả về giá trị nào nếu khai báo kiểu trả về là void

*Câu 17 : Khi nào thì phương thức khởi dựng được gọi?*

Phương thức khởi dụng là một phương thức sẽ được gọi thực hiện khi chúng ta tạo một đối tượng. Chúng được gọi khi tạo một đối tượng.

**Kế thừa, đa hình**

*Câu 22 : Có cần thiết phải chỉ định từ khóa* ***override*** *trong phương thức phủ quyết của lớp dẫn xuất hay không?*

*Câu 23 : Lớp trừu tượng là thế nào? Có thể tạo đối tượng cho lớp trừu tượng hay không?*

*Câu 24 : Có phải khi tạo một lớp thì phải kế thừa từ một lớp nào không?*

*Câu 25 : Lớp lồng bên trong một lớp là như thế nào?*

*Câu 26 : Có thể kế thừa từ một lớp cơ sở được viết trong ngôn ngữ khác ngôn ngữ C#?*

*Câu 27 : Khái niệm đa hình là gì? Khi nào thì cần sử dụng tính đa hình?*

*Câu 28 : Từ khóa new được sử dụng làm gì trong các lớp?*

*Câu 29 : Lớp cô lập là gì? Có thể khai báo protected cho các thành viên của nó được không?*

**Cấu trúc**

*Câu 33 : Sự khác nhau giữa cấu trúc và lớp?*

Trong C#, ta có thể xây dựng véc tơ theo kiểu struct hoặc class. Chọn lựa kiểu dữ liệu tùy thuộc vào yêu cầu mô tả dữ liệu. Ta có so sánh giữa 2 kiểu dữ liệu này như sau:

|  |  |
| --- | --- |
| Struct | Class |
| - Là kiểu dữ liệu trực tiếp, hay struct trực tiếp chứa dữ liệu.  - Sử dụng struct sẽ giảm tải cho garbage collector  - Được lưu trong stack  - Truyền tham số kiểu struct nhanh hơn kiểu class  - Thích hợp với cấu trúc dữ liệu nhỏ, sử dụng nhiều kiểu dữ liệu dựng sẵn | - Là kiểu đối tượng, tức là chỉ chứa tham chiếu đến đối tượng  - Luôn được hủy nhờ garbage collector  - Được lưu trong heap  - Thích hợp với các cấu trúc dữ liệu phức tạp và lập trình phức tạp |

Microsoft đã khuyến khích dùng kiểu struct để xây dựng các kiểu dữ liệu nhỏ, và kích thước thường sẽ nhỏ hơn 16 byte

*Câu 34 : Cấu trúc được lưu trữ ở đâu?*

Cấu trúc được lưu trữ trong stack

*Câu 35 : Truyền cấu trúc cho một phương thức dưới hình thức nào?*

Do là kiểu giá trị nên khi truyền một đối tượng cấu trúc cho một phương thức thì nó được truyền dưới dạng tham trị chứ không phải tham chiếu.

*Câu 36 : Chúng ta có thể khởi tạo giá trị ban đầu cho các biến thành viên của nó như bên dưới được không? Nếu không được tại sao?*

***struct myStruct***

***{***

***private int mNum = 100;***

***....***

***}***

Không vì bên trong 1 Struct, fields khi khai báo không thể được khởi tạo trừ khi chúng được khai báo như static, const

*Câu 37 : Sự khác nhau giữa kiểu dữ liệu tham chiếu và kiểu dữ liệu giá trị?*

Sự khác nhau giữa kiểu dữ liệu tham chiếu và kiểu dữ liệu giá trị là giá trị vào trong bộ nhớ

+ kiểu giá trị: sẽ được lưu giữ kích thước thật trong bộ nhớ đã cấp phát là stack.

+ kiểu tham chiếu: địa chỉ của kiểu dữ liệu tham chiếu thì được lưu trong stack nhưng đối tượng thật sự thì lưu trong bộ nhớ heap.

*Câu 38 : Sự khác nhau giữa bộ khởi dựng của cấu trúc và bộ khởi dựng của lớp?*

**Mảng, chỉ mục (index)**

*Câu 43 : Điều phân biệt giữa mảng và các thành phần bên trong một mảng?*

*Câu 44 :Một lớp có bộ chỉ mục khác một mảng như thế nào?*

*Câu 45 :Cách tạo ra mảng đa chiều không cùng kích thước?*

Khi chúng ta tạo một mảng đa chiều kích thước khác nhau thì chúng ta khai báo số dòng trong mảng trước. Sau đó với mỗi dòng sẽ giữ một mảng, có kích thước bất kỳ. Những mảng này được khai báo riêng. Sau đó chúng ta khởi tạo giá trị các thành phần trong những mảng bên trong.  
Trong mảng này, mỗi chiều là một mảng một chiều. Để khai báo mảng đa chiều có kích thước khác nhau ta sử dụng cú pháp sau, khi đó số ngoặc chỉ ra số chiều của mảng:  
<kiểu dữ liệu> [] [] ...  
Ví dụ, chúng ta có thể khai báo mảng số nguyên hai chiều khác kích thước tên myJaggedArray như sau:  
int [] [] myJaggedArray;  
Chúng ta có thể truy cập thành phần thứ năm của mảng thứ ba bằng cú pháp: myJaggedArray[2][4].

*Câu 46 :Hãy cho biết sự khác nhau giữa hai cách gọi Arr[i][j] và Arr[i, j]?*

Khi chúng ta truy cập các thành phần của mảng kích thước bằng nhau, chúng ta đặt tất cả các chỉ mục của các chiều vào trong cùng dấu ngặc vuông:  
rectangularArray[i,j]  
Tuy nhiên với mảng có kích thước khác nhau ta phải để từng chỉ mục của từng chiều trong dấu ngoặc vuông riêng:  
jaggedArray[i][j]

*Câu 47 :Có cách nào tạo một mảng mà không cần khai báo trước kích thước của mảng?*

Không có cách nào

*Câu 48 :Nếu mảng có 31 phần tử thì dung lượng của đối tượng ArrayList là bao nhiêu? Trường hợp có 33 phần tử?*

*Câu 49 :Hàng đợi là gì? Chúng được sắp xếp theo kiểu thứ tự nào? Ứng dụng của hàng đợi ?*

Hàng đợi là một tập hợp trong đó có thứ tự vào trước và ra trước (FIFO). Tương tự như là  
những người mua vé tàu, họ xếp thành một hàng, người nào vào trước thì sẽ mua trước và ra  
trước. Hàng đợi là kiểu dữ liệu tốt để quản lý những nguồn tài nguyên giới hạn. Ví dụ, chúng  
ta muốn gởi thông điệp đến một tài nguyên mà chỉ xử lý được duy nhất một thông điệp một  
lần. Khi đó chúng ta sẽ thiết lập một hàng đợi thông điệp để xử lý các thông điệp theo thứ tự  
đưa vào.

Cấu trúc dữ liệu hàng đợi có nhiều ứng dụng: khử [đệ qui](http://vi.wikipedia.org/wiki/%C4%90%E1%BB%87_quy), tổ chức lưu vết các quá trình [tìm kiếm theo chiều rộng](http://vi.wikipedia.org/wiki/T%C3%ACm_ki%E1%BA%BFm_theo_chi%E1%BB%81u_r%E1%BB%99ng) và [quay lui](http://vi.wikipedia.org/wiki/Quay_lui_%28khoa_h%E1%BB%8Dc_m%C3%A1y_t%C3%ADnh%29), [vét cạn](http://vi.wikipedia.org/w/index.php?title=V%C3%A9t_c%E1%BA%A1n&action=edit&redlink=1), tổ chức quản lý và phân phối [tiến trình](http://vi.wikipedia.org/wiki/Ti%E1%BA%BFn_tr%C3%ACnh) trong các [hệ điều hành](http://vi.wikipedia.org/wiki/H%E1%BB%87_%C4%91i%E1%BB%81u_h%C3%A0nh), tổ chức [bộ đệm bàn phím](http://vi.wikipedia.org/w/index.php?title=B%E1%BB%99_%C4%91%E1%BB%87m_b%C3%A0n_ph%C3%ADm&action=edit&redlink=1).

*Câu 50 :Ngăn xếp là gì? Chúng được sắp xếp theo kiểu thứ tự nào? Ứng dụng của ngăn xếp?*

Ngăn xếp là một tập hợp mà thứ tự là vào trước ra sau hay vào sao ra trước (LIFO), tương  
như một chồng đĩa được xếp trong nhà hàng. Đĩa ở trên cùng tức là đĩa xếp sau thì được lấy  
ra trước do vậy đĩa nằm dưới đáy tức là đĩa đưa vào đầu tiên sẽ được lấy ra sau cùng.

Ứng dụng của stack  
Có rất nhiều ứng dụng của stack, sau đây ta chỉ xét một số ứng dụng nhỏ:  
1.Đảo ngược xâu.  
Ta sẽ nhập xâu vào stack, sau đó lấy lần lượt các phần tử của stack ra tại đỉnh của nó. Và do đó ta có xâu đã đảo ngược.  
2.Đổi một số n từ hệ cơ số thập phân sang hệ nhị phân.  
Ta sẽ lấy số dư n%2 lưu vào stack, và gán n=n/2.  
Đến khi nào n=1 thì cũng lưu vào stack.  
Ta lại lấy lần lượt các phần tử của stack -> xâu biểu diễn nhị phân của n.  
  
3.Thao tác trên stack tổng quát cùng với các thao tác duyệt stack , tìm kiếm trên stack có rất nhiều ứng dụng.