Trường ĐH Khoa Học Tự Nhiên Tp. Hồ Chí Minh
**TRUNG TÂM TIN HỌC**

# Lập trình iOS

## Bài 2. *Các kiểu dữ liệu cơ sở trong Objective – C (Phần 1)*

**Ngành Mạng và Thiết bị di động**

**2015**

# Nội dung

1. **Hệ thống lưu trữ dữ liệu**
   - Vấn đề về lưu trữ dữ liệu trong máy tính
   - Vấn đề về sử dụng dữ liệu
   - Vấn đề về đa dạng dữ liệu
2. **Các kiểu dữ liệu cơ sở**
3. **Phép tính toán trên các dữ liệu cơ sở**
4. **Foundation Framework & Các kiểu dữ liệu trong Objective-C**

# 1.1 Vấn đề về lưu trữ dữ liệu

- **Ứng dụng được xây dựa nhu cầu thực tế và cần lưu trữ thông tin để có thể truy xuất khi cần.**
  - Ví dụ: xây dựng ứng dụng quản lý học viên, cần lưu trữ thông tin học viên như họ tên, ngày tháng năm sinh…
- **Máy tính sử dụng đơn vị cơ bản nhất gọi là Bit, bao gồm hai giá trị cơ bản 0 và 1 tương ứng với tín hiệu điện tắt và bật.**
  - Ví dụ: khi thực thi ứng dụng, máy tính sẽ nhận được một chuỗi dữ liệu như sau: 000110101010001110100 1….
- **Đơn vị cao hơn Bit được gọi là Byte, với 1byte = 8bit.**
  - Giá trị thấp nhất của Byte: 00000000 = 0
  - Giá trị cao nhất của Byte: 11111111 = 255
  - => mỗi byte có thể lưu trữ 256 giá trị khác nhau.

# 1.2 Vấn đề về sử dụng dữ liệu

- **Ngôn ngữ máy nhị phân quá phức tạp cho lập trình viên khi sử dụng:**
  - Ví dụ để lưu kí tự 'A' ta cần truyền mã nhị phân: **01000001**
- **Kiểu dữ liệu cơ sở được sinh ra nhằm đơn giản hóa cách khai báo và sử dụng dữ liệu:**
  - Ví dụ để lưu số nguyên có giá trị bằng 1. Ta khai báo:
    - int i = 1;
    - Ý nghĩa:
      - **int** : kiểu dữ liệu cơ sở số nguyên.
      - **i** : được gọi là tên **Biến**, dùng để truy xuất giá trị khi cần.
      - **=** : phép tính thực hiện gán giá trị cho một **Biến**.
      - **1** : giá trị được lưu trữ bên dưới bộ nhớ cho **biến i**.
    - Cách hiểu: khai báo biến i với giá trị được lưu trữ bằng 1.

# 1.3 Vấn đề về đa dạng dữ liệu

- **Nhu cầu thực tế cần lưu trữ rất nhiều dạng dữ liệu khác nhau như: chuỗi, số nguyên, số thập phân… với kích thước khác nhau.**
  - Ví dụ để lưu trữ thông tin về đặt vé xem phim trong ứng dụng xem phim:
    - Tên phim: Hobbit – Desolation of Smaug (kiểu chuỗi)
    - Ngày chiếu: 03/01/2015 (kiểu ngày tháng)
    - Tiền vé: 135.000 (kiểu số)
- **Hệ thống các kiểu dữ liệu cơ sở được tạo ra và cấp phát không gian lưu trữ phù hợp cho từng kiểu.**
  - Kiểu dữ liệu số: lưu trữ các giá trị kiểu số.
    - Ví dụ: 123, 67.89…
  - Kiểu dữ liệu kí tự: lưu trữ giá trị một kí tự bất kì.
    - Ví dụ: ‘A’, ‘1’, ‘₹’, ‘®’…
  - Kiểu dữ liệu luận lý: lưu trữ một trong hai giá trị 0 hoặc 1.

# Nội dung

1. Hệ thống lưu trữ dữ liệu

2. **Các kiểu dữ liệu cơ sở**
   - Kiểu dữ liệu số
     - Kiểu dữ liệu số nguyên
     - Kiểu dữ liệu số thực
   - Kiểu dữ liệu kí tự
   - Kiểu dữ liệu luận lý

3. Phép tính toán trên các dữ liệu cơ sở

4. Các kiểu dữ liệu cơ sở trong Objective-C

# 2.1 Kiểu dữ liệu số

- **Kiểu dữ liệu số dùng để lưu trữ các dữ liệu số phục vụ tính toán từ các nhu cầu phát sinh thực tế.**
  - Ví dụ: Tính toán số tiền cần thanh toán khi tiến hành đặt 4 vé phim, với mỗi vé giá 135.000. Khi đó: tổng tiền = 4x135.000.
- **Dữ liệu số cần lưu trữ có thể là một con số có độ lớn bất kỳ, do đó cần nắm rõ bộ nhớ cấp phát cho từng kiểu.**

| Kiểu dữ liệu | Không gian lưu trữ |
|---|---|
| short | 2 byte |
| int | 4 byte |
| float | 4 byte |
| double | 8 byte |
| long | 8 byte |
| long long | 16 byte |
| long double | 16 byte |

# 2.1.1 Kiểu dữ liệu số nguyên

- Kiểu dữ liệu số nguyên được gán giá trị bằng một con số bao hàm cả có dấu và không dấu.
- Để có lưu trữ một dữ liệu số nguyên bất kỳ thay vì cấp phát chính xác số lượng byte tương ứng ta sẽ dựa trên mối tương quan của các kiểu dữ liệu.
  - short <= int <= long <= long long

# 2.1.1 Kiểu dữ liệu số nguyên

- **Trong một ít trường hợp liên quan đến thuật toán cần sự chính xác tuyệt đối khi đó cần sử dụng dạng dữ liệu đặc biệt của kiểu số nguyên.**
  - (u)int<n>_t: định nghĩa số nguyên có dấu hoặc không dấu có độ dài **n** tương ứng 1, 2, 4 hoặc 8 byte. (ký tự "u" có nghĩa là số không dấu (unsigned)).

```
int8_t aOneByteInt = 127;
uint8_t aOneByteUInt = 255;
int16_t aTwoByteInt = 32767;
uint16_t aTwoByteUInt = 65535;
int32_t aFourByteInt = 2147483647;
uint32_t aFourByteUInt = 4294967295;
int64_t anEightByteInt = 9223372036854775807;
uint64_t anEightByteUInt = 18446744073709551615;
```

# 2.1.1 Kiểu dữ liệu số nguyên

- (u)int_least<n>_t: định nghĩa số nguyên qui định kích thước nhỏ nhất
  - int_least8_t aTinyInt = 127;
  - uint_least8_t aTinyUInt = 255;
  - int_least16_t aMediumInt = 32767;
  - uint_least16_t aMediumUInt = 65535;
  - int_least32_t aNormalInt = 2147483647;
  - uint_least32_t aNormalUInt = 4294967295;
  - int_least64_t aBigInt = 9223372036854775807;
  - uint_least64_t aBigUInt = 18446744073709551615;
- (u)intmax_t: định nghĩa số nguyên qui định kích thước lớn nhất hệ thống có thể xử lý.
  - intmax_t theBiggestInt = 9223372036854775807;
  - uintmax_t theBiggestUInt = 18446744073709551615;

# 2.1.1 Kiểu dữ liệu số nguyên

- **Để truy xuất hiển thị dữ liệu số nguyên ta dùng kí hiệu %d cho số nguyên có dấu và %u cho số nguyên không dấu.**
  - Truy xuất có số nguyên có dấu và không dấu:

```
int anInt = -2147483648;
unsigned int anUnsignedInt = 4294967295;
NSLog(@"%d", anInt);
NSLog(@"%u", anUInt);
```

# 2.1.1 Kiểu dữ liệu số nguyên

- **Bảng định dạng truy xuất:**

| Kiểu dữ liệu | Định dạng truy xuất có dấu | Định dạng truy xuất không dấu |
|---|---|---|
| short | hd | hu |
| int | d | u |
| long | ld | lu |
| long long | lld | llu |

# 2.1.2 Kiểu dữ liệu số thực

- **Giống với kiểu dữ liệu số nguyên tuy nhiên có thể lưu trữ được các số thập phân không phân biệt có dấu và không dấu bao gồm ba định dạng:**
  - float <= double <= long double.

| Kiểu dữ liệu | Không gian lưu trữ | Định dạng truy xuất |
|---|---|---|
| float | 4 byte | f |
| double | 8 byte | f |
| long double | 16 byte | Lf |

# 2.1.2 Kiểu dữ liệu số thực

- **Để truy xuất số lượng chữ số trước và sau dấu chấm thập phân sử dụng định dạng <m>.<n>**
  - Ví dụ:

```
float aFloat = -21.096758;
NSLog(@"%8.2f", aFloat);
```

  - Kết quả in ra màn hình:
    - -21.09

# 2.2 Kiểu ký tự

- **Kiểu dữ liệu kí tự được sử dụng để lưu các giá trị bao gồm: chữ số, kí hiệu, chữ cái và một số biểu tượng khác.**
- **Kiểu dữ liệu kí tự được cấp phát duy nhất một byte để lưu trữ dữ liệu (tương đương 256 giá trị) và do đó chỉ được phép lưu một kí tự duy nhất.**
  - Ví dụ khai báo dữ liệu kiểu kí tự:
    - char c = 'A'; // Đúng
    - char c = 'ABC'; // Sai
- **Kiểu dữ liệu ký tự có thể khai báo có dấu và không dấu.**
  - Ví dụ:
    - char a = 'A'
    - unsigned char a = 98

# 2.2 Kiểu ký tự

- **Truy xuất giá trị của kiểu kí tự:**
  - %c : truy xuất giá trị có dấu
  - %hhu : truy xuất giá trị không dấu
  - %hhd : truy xuất thứ tự trong bảng mã ASCII
  - %s : truy xuất giá trị con trỏ kiểu kí tự

```
char aChar = 'A';
unsigned char anUChar = 255;

NSLog(@"%c", aChar);      // A
NSLog(@"%hhd", aChar);    // 65
NSLog(@"%hhu", anUChar); // 255
```

# 2.2 Kiểu ký tự

- **Vấn đề: cần lưu trữ thông tin là một chuỗi nhiều kí tự.**
  - Ví dụ: cần lưu trữ tên phim: "Hobbit 2 – Desolation of Smaug"
- **Thực hiện khai báo kiểu kí tự ở dạng "con trỏ", chuỗi lưu trữ sẽ được quản lý bởi "con trỏ" này.**
  - Ví dụ:

```
char* a = "Hobbit 2 - Desolation of Smaug";
NSLog(@"%s", a);
```

# 2.3 Kiểu luận lý

- **Kiểu luận lý (bool) được hiểu ở ngôn ngữ máy với hai giá trị 0 hoặc 1, ở môi trường lập trình bao gồm hai giá trị true và false.**
- **Các biến bool tường được dùng để kiểm tra giá trị của một đối tượng hoặc điều hướng trong các câu điều kiện.**
  - Ví dụ về cách khai báo và truy xuất:

```
bool isBool = true;

NSLog(@"%d", isBool);
```

  - Kết quả in ra màn hình:
    - 1

# Nội dung

1. Hệ thống lưu trữ dữ liệu

2. Các kiểu dữ liệu cơ sở

3. **Phép tính toán trên các dữ liệu cơ sở**
   - Toán tử tính toán
   - Toán tử so sánh
   - Toán tử Logic

4. Foundation Framewrok & Các kiểu dữ liệu cơ sở trong Objective-C

# 3.1 Toán tử tính toán

- **Toán tử tính toán được sử dụng trong các quá trình xử lý dữ liệu đưa các kết quả tính toán số học theo ý nghĩa thực thi.**
- **Phép tính toán cơ bản nhất bao gồm: 2 mệnh đề dữ liệu, một phép gán và một toán tử.**
  - Ví dụ: int i = 1 + 2;
  - Ý nghĩa:
    - Mệnh đề dữ liệu 1: **int i**
    - Mệnh đề dữ liệu 2: **1 + 2**
    - Toán tử sử dụng: **+**
    - Phép gán: **=**
  - Kết quả biến i khi in ra màn hình:
    - 3

# 3.1 Toán tử tính toán

## Các toán tử tính toán

| Toán tử tính toán | Ý nghĩa |
|---|---|
| + | Phép cộng |
| - | Phép trừ |
| * | Phép nhân |
| / | Phép chia |
| % | Phép chia lấy phần dư |

## Ví dụ:

```
NSLog(@"6 + 2 = %d",   6 + 2);     // 8
NSLog(@"6 - 2 = %d",   6 - 2);     // 4
NSLog(@"6 * 2 = %d",   6 * 2);     // 12
NSLog(@"6 / 2 = %d",   6 / 2);     // 3
NSLog(@"6 %% 2 = %d", 6 % 2);     // 0
```

# 3.1 Toán tử tính toán

- **Các phép tính toán được thực hiện theo độ ưu tiên phép tính toán số học và cơ chế từ trái qua phải.**
  - Ví dụ: int i = 3 * 4 – 6 / 2 % 3;
  - Kết quả: i = 12
- **Để ưu tiên các phép tính toán ta thực hiện "đóng ngoặc" các phép tính toán đó.**
  - Ví dụ: int i = 3 * 4 – 6 / ( 2 % 3 );
  - Kết quả: i = 9

# 3.1 Toán tử tính toán

- **Đối với phép tính cộng và trừ được bổ sung hai toán tử ++ và -- (chỉ sử dụng cho kiểu dữ liệu số)**
  - ++ : cho phép tăng một đơn vị
  - -- : cho phép giảm một đơn vị
- **Ví dụ:**

```
int soNguyen = 7;
NSLog(@"soNguyen + 1 = %d", ++soNguyen);     // 8

float soThuc = 11.35;
NSLog(@"soNguyen - 1 = %.2f", --soThuc);     // 10.35
```

- **Lưu ý: vị trí đặt các toán tử trước và sau biến dữ liệu có ý nghĩa khác nhau, ví dụ:**

  ++soNguyen: tăng giá trị số nguyên sau đó ghi kết quả (ví dụ trên)

  soNguyen++ : ghi số nguyên sau đó tăng giá trị

# 3.1 Toán tử tính toán

- Có thể thực hiện tính toán và gán giá trị trực tiếp thông qua các toán tử rút gọn.

| Toán tử tính toán | Ví dụ | Ý nghĩa |
|---|---|---|
| + | x += y | x = x + y |
| - | x -= y | x = x - y |
| * | x *= y | x = x * y |
| / | x /= y | x = x / y |
| % | x %= y | x = x % y |

- Phép rút gọn có thể hiểu là biến ở mệnh đề dữ liệu 1 sẽ tăng hoặc giảm một lượng bằng với giá trị của biến ở mệnh đề dữ liệu 2.

# 3.2 Toán tử so sánh

- Toán tử so sánh được sử dụng trong các trường hợp kiểm tra giá trị giữa 2 biến cùng kiểu dữ liệu. Kết quả trả về được đại diện bằng một biến bool (true hoặc false)

| Toán tử so sánh | Ý nghĩa |
|---|---|
| == | Bằng (giá trị) |
| > | Lớn hơn |
| >= | Lớn hơn hoặc bằng |
| < | Nhỏ hơn |
| <= | Nhỏ hơn hoặc bằng |
| != | Không bằng (giá trị) |

# 3.3 Toán tử Logic

- Toán tử Logic được dùng kết hợp với các cấu trúc điều khiển để kiểm tra giá trị biến hoặc tập hợp mệnh đề.

| Toán tử logic | Ví dụ | Ý nghĩa | Kết quả |
|---|---|---|---|
| ! | !a | Phủ định giá trị biến | Trả về true nếu giá trị khác a và ngược lại |
| && | <mệnh đề 1><br>&&<br><mệnh đề 2> | Kiểm tra giá trị mệnh đề 1 kết hợp giá trị mệnh đề 2 | Trả về true nếu cả hai mệnh đề có giá trị là true |
| \|\| | <mệnh đề 1><br>\|\|<br><mệnh đề 2> | Kiểm tra giá trị mệnh đề 1 kết hợp giá trị mệnh đề 2 | Trả về false nếu một trong hai mệnh đề so sánh trả về false |

# Nội dung

1. Hệ thống lưu trữ dữ liệu
2. Các kiểu dữ liệu cơ sở
3. Phép tính toán trên các dữ liệu cơ sở
4. **Foundation Framework & Các Các kiểu dữ liệu cơ sở trong Objective - C**
   - Foundation Framework
   - Các kiểu dữ liệu cơ sở Objectice-C
     - NSInteger
     - NSString

# 4.1 Foundation FrameWork

- **Bộ Foundation Framework định nghĩa một tập cơ sở các lớp trong Objective-C bổ sung các đối tượng dữ liệu tiện ích bên cạnh các kiểu dữ liệu cơ sở.**
- **Mục đích sử dụng  chính:**
  - Cung cấp các lớp chức năng cơ bản (chuyển đổi, lưu đối tượng…)
  - Xây dựng ứng dụng dễ hơn trong cấp phát và thu hồi bộ nhớ
  - Hỗ trợ các định dạng chuỗi, duy trì và phân chia đối tượng
  - Nâng cao tính linh hoạt trong xây dựng tầng hệ thống độc lập
    - True: Đối tượng được hiển thị
    - False: Đối tượng bị ẩn
- **Lớp NSObject là làm gốc trong bộ Foundation Framework**

# 4.1 Foundation FrameWork

- **Bộ Foundation Framework được chia thành các lớp riêng rẻ, mỗi lớp được gom nhóm dựa trên chức năng, bao gồm:**
  - **Lưu trữ dữ liệu**: NSArray, NSDictionary, NSSet …
  - **Chuỗi và văn bản**: NSCharacterSet, NSString, NSScanner…
  - **Ngày giờ**: NSDate, NSTimeZone, NSCalendar…
  - **Tính toán thời gian và điều phối ứng dụng**: NSNotification, NSNotificationCenter, NSTimer,…
  - **Tạo và hủy đối tượng:** NSAutoreleasePool.
  - **Nén và giải nén đối tượng**: NSCoder và các lớp con của nó.
  - **Operating-system services**: NSFileManager, NSThread, NSProcessInfo,…
  - **Hệ thống tải URL:** NSURL, NSURLRequest, NSURLResponse,…

# 4.2.1 NSInteger

- Kiểu dữ liệu số nguyên trong Objective-C, cho phép lưu trữ các giá trị số nguyên (không dấu - NSUInteger).
- Được khai báo kiểu struct, không có hàm cấp phát.
- Được sử dụng để tăng tính hiệu quả của ứng dụng trên các kiến trúc khác nhau.
  - int: phân biệt về cấp phát khi chạy trên nền 32bit – 64bit
  - NSInteger: không phân biệt cấp phát trên các nền khác nhau.
- Được sử dụng rộng rãi trong các phương thức trả về giá trị số nguyên trong bộ Cocoa & CocoTouch (TableView, PageControl).
- Dễ dàng chuyển đổi sang các kiểu dữ liệu trong Objective-C (NSNumber, NSValue, NSString…)

# 4.2.2 NSString

- **NSString là một kiểu dữ liệu dạng lớp đối tượng, dùng để hiển thị đoạn văn bản trong ứng dụng Objective-C. Bên cạnh việc cung cấp một chuỗi đóng gói hướng đối tượng, NSString còn cung cấp nhiều phương thức mạnh mẽ để tìm kiếm và thao tác với nội dung. Ngoài ra nó còn hỗ trợ Unicode.**
- **Các cách tạo một chuỗi kiểu NSString.**

| | |
|---|---|
| Tạo bằng cách gán với chuỗi. | NSString *module= @"bài 2"; |
| Tạo đối tượng theo định dạng | NSString *say=<br>[NSString stringWithFormat:@"Xin chào đây là %@", module]; |

# 4.2.2 NSString

- **So sánh chuỗi**

| | |
|---|---|
| So sách 2 chuỗi bằng nhau. | `NSString *traiCay = @"Sầu riêng";`<br>`if ([traiCay isEqualToString:@"Sầu riêng"]) {`<br>`  NSLog(@"Đây là trái sầu riêng");`<br>`}`<br>`//Đây là trái sầu riêng` |

# 4.2.2 NSString

- **So sánh hai chuỗi khác nhau dùng compare.**

So sánh chuỗi bằng NSComparisonResult. Kết quả trả về sẽ là một enum NSComparisonResult bao gồm: NSOrderedSame, NSOrderedAscending, NSOrderedDecending

```
NSString *tenToi=@"Hiền";
NSString *tenBanToi=@"Minh";
NSComparisonResult ketqua = [tenToi
compare:tenBanToi];
if (ketqua == NSOrderedAscending) {
  NSLog(@"Tên tôi đứng trước tên bạn
tôi");
} elseif (ketqua == NSOrderedSame) {
  NSLog(@"Tên hai đứa tôi giống nhau.");
} elseif (ketqua == NSOrderedDescending)
{
  NSLog(@"Tên tôi đứng sau tên bạn tôi");
}

// Tên tôi đứng trước tên bạn tôi
```

# 4.2.2 NSString

## ❑ Kiểm tra chuỗi

| | |
|---|---|
| Dùng hasPrefix so sánh với cụm đầu của chuỗi. | `NSString *car = @"Porsche Carrera";`<br>`if ([car hasPrefix:@"Porsche"]) {`<br>`  NSLog(@"Đây là xe Porsche");`<br>`}//Đây là xe Porsche` |
| Dùng hasSuffix so sánh với cụm cuối của chuỗi. | `if ([car hasSuffix:@"Carrera"]) {`<br>`  NSLog(@"Đây là xe Carrera");`<br>`} else {`<br>`  NSLog(@"Đây là xe Porsche");`<br>`}//Đây là xe Porsche` |

# 4.2.2 NSString

- Ta có thể nối chuỗi bằng các phương thức sau:

```
NSString *ten = @"Thùy";
NSString *ho = @"Nguyễn";
//Nối chuỗi trực tiếp
NSString *hoTen = [ho stringByAppendingString: ten];
NSLog(@"%@",hoTen); //NguyễnThùy
//Nối chuỗi theo định đạng
hoTen = [ho stringByAppendingFormat:@" %@", ten];
NSLog(@"%@",hoTen); //Nguyễn Thùy
//Nối chuỗi kèm theo dấu ‘/’ thường dùng cho các loại đường dẫn
hoTen = [ho stringByAppendingPathComponent:ten];
NSLog(@"%@",hoTen); //Nguyễn/Thùy
//Nối chuỗi kèm theo dấu ‘.’ thường dùng cho các loại tên tập
tin
hoTen = [ho stringByAppendingPathExtension:ten];
NSLog(@"%@",hoTen); //Nguyễn.Thùy
```

# 4.2.2 NSString

- Ta có thể tìm chuỗi trong chuỗi bằng NSRange

```
NSString *chuoi = @"Trung tâm tin học Đại Học Khoa Học Tự
Nhiên";
NSRange ketqua = [chuoi rangeOfString:@"họ"];
if (ketqua.location == NSNotFound) {
  NSLog(@"Không tìm thấy");
} else {
  NSLog(@"Chữ 'họ' bắt đầu tại vị trí thứ %lu và có chiều dài
là %lu", ketqua.location, ketqua.length);
  //Chữ 'họ' bắt đầu tại vị trí thứ 14 và có chiều dài là 2"
}
```

# 4.2.2 NSString

- Trong nhiều trường hợp ta cần chia chuỗi thành nhiều chuỗi nhỏ hơn. Có 3 phương thức hỗ trợ cắt chuỗi như

```
NSString *chuoi = @"Trung tâm tin học Đại Học Khoa Học Tự
Nhiên";
//Cắt chuỗi từ đầu đến vị trí thứ 17
NSLog(@"%@", [chuoi substringToIndex:17]); //Trung tâm tin học
//Cắt chuỗi từ vị trí thứ 19 đến cuối chuỗi
NSLog(@"%@", [chuoi substringFromIndex:19]); //Đại Học Khoa Học
Tự Nhiên
NSRange range = NSMakeRange(19, 7);
//Cắt chuỗi từ vị trí thứ 19 và lấy 7 ký tự
NSLog(@"%@", [chuoi substringWithRange:range]); //Đại Học
```

# 4.2.2 NSString

- Ngoài ra ta còn có thể cắt chuỗi thành một mảng danh sách chuỗi:

```
NSString *danhSachTen = @"Thùy, Kiều, Minh, Tâm";
NSArray *mangChuaTen = [danhSachTen
componentsSeparatedByString:@","];
for (int i = 0; i < mangChuaTen.count; i ++) {
  NSLog(@"%@", [mangChuaTen objectAtIndex:i]); //In ra một
phần tử trong mảng
}
//Thùy
//Kiều
//Minh
//Tâm
```

# 4.2.2 NSString

- NSString hỗ trợ 3 phương thức thay đổi hoa/thường cho chuỗi.

```
NSString *chuoi = @"Trung tâm tin học Đại Học Khoa Học Tự Nhiên";
NSLog(@"%@", [chuoi lowercaseString]); //In ra chuỗi ký tự in thường
//trung tâm tin học đại học khoa học tự nhiên
NSLog(@"%@", [chuoi uppercaseString]); //In ra chuỗi ký tự in hoa
//TRUNG TÂM TIN HỌC ĐẠI HỌC KHOA HỌC TỰ NHIÊN
NSLog(@"%@", [chuoi capitalizedString]); //In ra chuỗi in hoa chữ cái đầu
//Trung Tâm Tin Học Đại Học Khoa Học Tự Nhiên
```

# 4.2.3 NSNumber

- Lớp NSNumber chứa các kiểu dữ liệu số trong C như bool, int, float, double,… Với mỗi loại dữ liệu sẽ có phương thức khởi tạo riêng

```
NSNumber *aBool = [NSNumber numberWithBool:NO];
NSNumber *aChar = [NSNumber numberWithChar:'z'];

NSNumber *aBool = @NO;
NSNumber *aChar = @'z';

NSLog(@"%@", [aBool boolValue] ? @"YES" : @"NO");
NSLog(@"%c", [aChar charValue]);
```

# 4.2.3 NSNumber

- So sánh: Tương tự như NSString, ta có thể sử dụng phương thức isEqualToNumber hoặc compare để so sánh giá trị của 2 số.

```
NSNumber *tuoiToi = @27;
NSNumber *tuoiBanToi = @27U;
if ([tuoiToi isEqualToNumber:tuoiBanToi]) {
  NSLog(@"Hai đứa tôi bằng tuổi");
} // Hai đứa tôi bằng tuổi
NSNumber *tuoiToi = @20;
NSNumber *tuoiBanToi = @30;
if ([tuoiToi compare: tuoiBanToi] == NSOrderedDescending) {
  NSLog(@"Tuổi tôi lớn hơn tuổi bạn tôi");
}else{
  NSLog(@"Tuổi bạn tôi lớn hơn tuổi tôi");
} //Tuổi bạn tôi lớn hơn tuổi tôi
```

# 4.2.4 NSDate

- **Lấy ngày:**

| | |
|---|---|
| Lấy ngày hiện tại | `NSDate *hienTai = [NSDate date];`<br>`//Hiện tại: 2015-05-20 02:50:50 +0000` |
| Lấy ngày có khoảng cách với một ngày định sẵn | `NSTimeInterval motTuan = 7 * 24 * 60 * 60;`<br>`NSDate *tuanTruoc = [NSDate dateWithTimeInterval:-motTuan sinceDate: hienTai];`<br>`NSDate *tuanSau = [NSDate dateWithTimeInterval: motTuan sinceDate: hienTai];` |
| So sánh ngày với một ngày xác định khác. | `NSDate *ngayNhoHon = [hienTai earlierDate: tuanSau]; //là ngày hienTai`<br>`NSDate *ngayLonHon = [hienTai laterDate: tuanSau]; //là ngày tuanSau`<br>`NSLog(@"%@ nhỏ hơn %@", ngayNhoHon, ngayLonHon);` |

# 4.2.4 NSDate

- So sánh: Tượng tự các kiểu dữ liệu trước, đối tượng NSDate cũng có các phương thức để so sánh như isEqualToDate và compare

```
NSComparisonResult ketQua = [hienTai compare:tuanSau];
if (ketQua == NSOrderedAscending) {
  NSLog(@"hienTai < tuanSau");
} elseif (ketQua == NSOrderedSame) {
  NSLog(@"hienTai == tuanSau");
} elseif (ketQua == NSOrderedDescending) {
  NSLog(@"hienTai > tuanSau");
}
```

# Thảo luận