

বাংলায়

iOS অ্যাপ ও গেম ডেভেলপমেন্ট

Objective-C

Swift

Xcode



iOS7, iOS8



অবজেক্টিভ-সি ও সুইফ্ট লার্নিং ডাটা ও গ্রাফিক্যাল অ্যাপ তৈরি 2D, 3D গেম ডেভেলপমেন্ট
অ্যাপ স্টোর সাবমিট অ্যাপ্লিকেশন থেকে রেভিনিউ এর বিস্তারিত অ্যাপ প্রমোশন

সূচিপত্র

ভূমিকা	0
অবজেক্টিভ-সি ব্যাসিক	1
প্রাথমিক ধারণা এবং Xcode	1.1
Objective-C এর ভিতরে আমাদের প্রিয় C	1.2
ফাংশন ও এর ব্যবহার	1.3
ক্লাস ও অবজেক্ট	1.4
প্রোপার্টি ও এর অ্যাট্রিবিউটের ব্যবহার	1.5
মেথড ও এর বিস্তারিত	1.6
প্রোটোকল	1.7
ক্যাটাগরি	1.8
ব্লক ও এর ব্যবহার	1.9
এক্সেপশন ও এর হ্যান্ডেলিং	1.10
অ্যাপ ডেভেলপমেন্ট	2
টুলস সেটআপ এবং একটি সাধারণ হ্যালো ওয়ার্ল্ড অ্যাপ তৈরি	2.1
iOS অ্যাপে ব্যাসিক ইনপুট আউটপুট ও কিবোর্ড হ্যান্ডেলিং	2.2
আরও চ্যাপ্টার আসছে ...	2.3
সুইফট ব্যাসিক	3
সুইফট – অ্যাপলের নতুন চমক, পরিচিতি ও অন্যান্য	3.1
স্ট্রিং ও ক্যারেকটার টাইপ ডারিয়েবল	3.2
কালেকশনস (অ্যারে ও ডিকশনারী), ইনুমারেশন ও ক্লোজার	3.3
গেম ডেভেলপমেন্ট	4
অ্যাপ্লিকেশন সাবমিট, প্রচার ও সাফল্য	5
অতিরিক্তঃ কিছু সাধারণ প্রশ্ন ও উত্তর	6
অবজেক্টিভ-সি (Objective-C) নাকি সুইফট (Swift)?	6.1

 Like  Share 61 people like this. [Sign Up](#) to see what your friends like.

[কোর্স এর মূল পাতা](#) | [HowToCode মূল সাইট](#) | [সবার জন্য প্রোগ্রামিং ব্লগ](#) | [পিডিএফ ডাউনলোড](#)

iOS অ্যাপ ও গেম ডেভেলপমেন্ট

 [gitter](#)  join chat

10

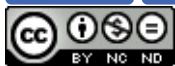
বাংলায় অবজেক্টিভ-সি, সুইফট এবং আইওএস অ্যাপ ও গেম ডেভেলপমেন্ট শেখার জন্য উপযুক্ত একটি কোর্স

ওপেন সোর্স

এই বইটি মূলত স্বেচ্ছাশ্রমে লেখা এবং বইটি সম্পূর্ণ ওপেন সোর্স। এখানে তাই আপনিও অবদান রাখতে পারেন লেখক হিসেবে। আপনার কন্ট্রিবিউশান গ্রহীত হলে অবদানকারীদের তালিকায় আপনার নাম যোগ করে দেওয়া হবে।

এটি মূলত একটি [গিটহাব রিপোজিটোরি](#) যেখানে এই বইয়ের আর্টিকেল গুলো মার্কডাউন ফরম্যাটে লেখা হচ্ছে। রিপোজিটোরিটি ফর্ক করে পুল রিকুয়েস্ট পাঠানোর মাধ্যমে আপনারাও অবদান রাখতে পারেন।

 Like 46  Share



This work is licensed under a [Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License](#).

এই সেকশনে থাকছে

- প্রাথমিক ধারণা এবং Xcode
- Objective-C এর ভিতরে আমাদের প্রিয় C
- ফাংশন ও এর ব্যবহার
- ক্লাস ও অবজেক্ট
- প্রোপার্টি ও এর অ্যাট্রিবিউটের ব্যবহার
- মেথড ও এর বিস্তারিত
- প্রোটোকল
- ক্যাটাগরি
- ব্লক ও এর ব্যবহার
- এক্সেপশন ও এর হ্যান্ডেলিং

প্রাথমিক ধারণা এবং Xcode

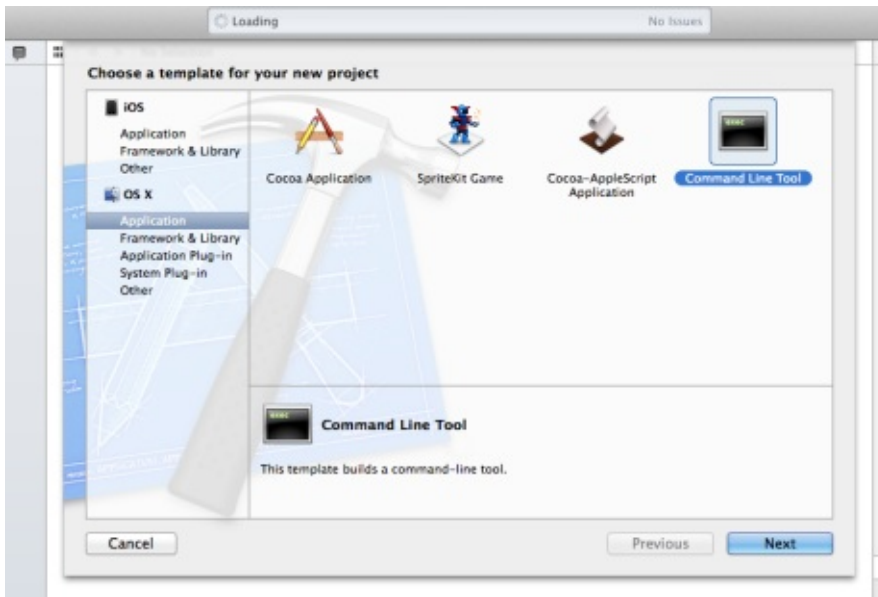
ভূমিকাঃ অবজেক্টিভ-সি একটি রিফ্লেক্টিভ, অবজেক্ট ওরিয়েন্টেড প্রোগ্রামিং ভাষা যাতে সি ভাষার সাথে স্মলটকের মেসেজ আদান পদ্ধতির সম্মিলন ঘটেছে। বর্তমানে এটি মূলত ম্যাক ওএস এক্স, আইওএস ও জিএনইউ স্টেপ সিস্টেমে ব্যবহৃত হয়। এই তিনটি সিস্টেম নির্মিত হয়েছে ওপেনস্টেপ স্ট্যান্ডার্ডের ওপর ভিত্তি করে যা নেত্রটস্টেপ, ওপেনস্টেপ ও কোকোয়া ফ্রেমওয়ার্কের প্রধান ভাষা। সাধারণ অবজেক্টিভ-সি প্রোগ্রাম অবশ্য এই ফ্রেমওয়ার্ক ব্যবহার করে না, বরং অবজেক্টিভ-সি কম্পাইলার সমৃদ্ধ জিসিসি সমর্থিত যেকোন সিস্টেমেই চলে। - [উইকি](#)

Objective-C হচ্ছে iOS এবং OSX অপারেটিং সিস্টেমের জন্য ন্যাটিভ প্রোগ্রামিং ল্যাঙ্গুয়েজ। এটা একটা কম্পাইল্ড ল্যাঙ্গুয়েজ। অর্থাৎ পুরো কোড আগে কম্পাইল হয়ে তারপর সেখান থেকে বিন্দু এবং রান হয়। Objective-C তৈরি করা হয়েছে C প্রোগ্রামিং ল্যাঙ্গুয়েজকে অবজেক্ট ওরিয়েন্টেড ফিচার দিয়ে। C এর তুলনায় Objective-C এর সিনট্যাক্সকে আরও বেশি হিউম্যান রিডেবল মনে করা হয়।

তুলনামূলক সিম্পল সিনট্যাক্স এর এই ল্যাঙ্গুয়েজ এর পিছনে আরও অবদান রয়েছে বিশাল বিস্তৃত কিছু লাইব্রেরীর। যদিও ল্যাঙ্গুয়েজের শেখার দৃষ্টিকোণ থেকে দেখতে গেলে ওই লাইব্রেরীগুলো সম্পর্কে এখনি জানা বাধ্যতামূলক নয় কিন্তু কিছুটা আইডিয়া রাখা ভালো। অনেক লাইব্রেরীর মধ্যে Apple এর Cocoa এবং Cocoa Touch ফ্রেমওয়ার্ক হচ্ছে বহুল পরিচিত। এগুলো কিছু API ডিফাইন করে যা দিয়ে যথাক্রমে OSX এবং iOS এর জন্য অ্যাপ্লিকেশন তৈরি করা সহজ হয়ে যায়। Foundation, UIKit, AppKit, CoreData QuartzCore, AVFoundation হচ্ছে তেমন কিছু ফ্রেমওয়ার্ক। Objective-C তে পারদর্শী হবার পর উপরে আলোচিত টুলস গুলো হচ্ছে iOS এবং OSX এর জন্য অ্যাপ তৈরির ক্ষেত্রে প্রয়োজনীয় জিনিষ। আপাতত এই ল্যাঙ্গুয়েজ নিয়ে ঘাটা ঘাটি করতে আমরা শুধুমাত্র Foundation ফ্রেমওয়ার্ক ব্যবহার করতে পারি।

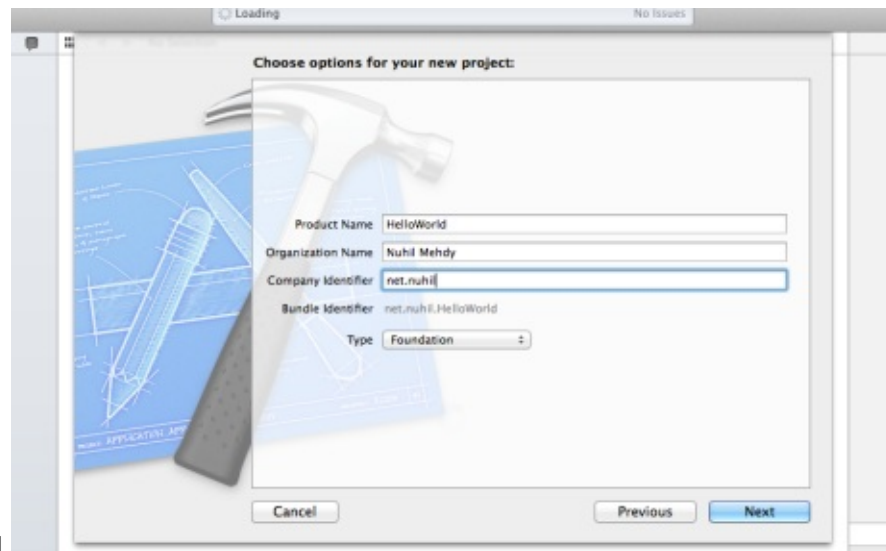
Xcode: যদিও Objective-C কোড কম্পাইল করার জন্য আরও অনেক পথ আছে কিন্তু আমরা এই সিরিজে শুরু থেকেই Xcode ব্যবহার করবো কারণ শেষ নাগাদ iOS, OSX অ্যাপ ডেভেলপ করাই আমাদের উদ্দেশ্য এবং তার জন্য Xcode এর সাথে পরিচিত থাকা অবশ্যই উপকারী। Xcode হচ্ছে Apple এর একটি IDE (Integrated Development Environment). তাছাড়াও এটা একটা কোড এডিটর, ডিবাগার টুল, অ্যাপ এর GU ইন্টারফেস বিন্ডার এবং এর সাথে সিমুলেটরও আছে।

Objective-C প্র্যাকটিসের উপযোগী অ্যাপ/এনডায়নমেন্ট তৈরিঃ iOS এবং OSX অ্যাপ তৈরি শুরু করার জন্য Xcode আমাদেরকে অনেক রকম টেমপ্লেট করে দিতে পারে। আমরা আমাদের এই টিউটোরিয়াল সিরিজের স্বার্থে, Command Line Tool টাইপের অ্যাপ টেমপ্লেট ব্যবহার করবো। এতে করে আমরা iOS/OSX অ্যাপ তৈরির জন্য অন্যসব টুলস থেকে দূরে থাকতে পারবো এবং শুধুমাত্র Objective-C সম্পর্কে জানার জন্য যতটুকু পরিবেশ দরকার সেটাই তৈরি করে আগাবো। Xcode এর File->New->Project এ গেলে নিচের মত উইন্ডো আসবে।



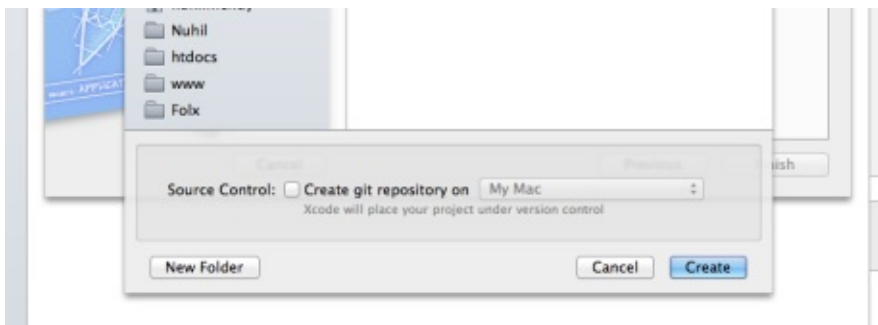
সেখানকার বাম পাস থেকে OSX

: Application সিলেক্ট করে ডান পাস থেকে Command Line Tool সিলেক্ট করে Next করলে নিচের মত



আরেকটি উইন্ডো আসবে।

এখানে অ্যাপ এর ব্যাসিক কিছু সেটিংস কনফিগার করা হয়। Product Name এবং Organization Name এ যা খুশি দিতে পারেন এবং Company Identifier এ edu.self দিতে পারেন private use identifier হিসেবে, যদি আপনার দেয়ার মত কোন ইউনিক রিভার্স ডোমেইন না থাকে। Type ফিল্ড থেকে Foundation সিলেক্ট করুন কারন আমরা সামনে এই ফ্রেমওয়ার্কে ডিফাইন করা বেশ কিছু ক্লাসের ব্যবহার পাবো। এখন Next ক্লিক করে প্রজেক্ট শেড লোকেশন দেখিয়ে দিন।



Create বাটনে ক্লিক করার পর Xcode এর বাম পাশের ন্যাভিগেটর-এ main.m ফাইল সহ আরও কিছু ফাইল ফোল্ডার দেখতে পাবেন। আপাতত এই main.m ফাইলটাই গুরুত্বপূর্ণ। কারণ iOS, OSX এমনকি Command Line Tool টাইপের অ্যাপ রান করার সময় এখান থেকেই ঘটনা শুরু হয় :) আর হ্যাঁ, .m হচ্ছে Objective-C এর সোর্স ফাইলের এক্সটেনশন।

main() ঘটনাঃ আগেই বলা হয়েছে, main() ফাংশন যেকোনো রকম Objective-C অ্যাপ এর রুট হিসেবে কাজ করে। Xcode প্রায় সব সবরকম অ্যাপ টেমপ্লেট তৈরি করার সময় তার জন্য একটি main.m ফাইল বানায় যেখানে একটি main() ফাংশন ডিফাইন করা থাকে। ফাইলটি খুললে নিচের মত কোড দেখা যাবেঃ

```
#import

int main(int argc, const char * argv[])
{

    @autoreleasepool {

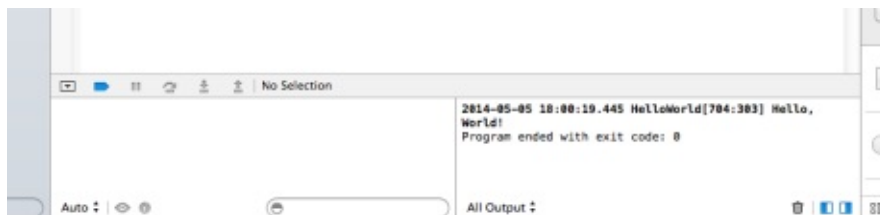
        // insert code here...
        NSLog(@"Hello, World!");

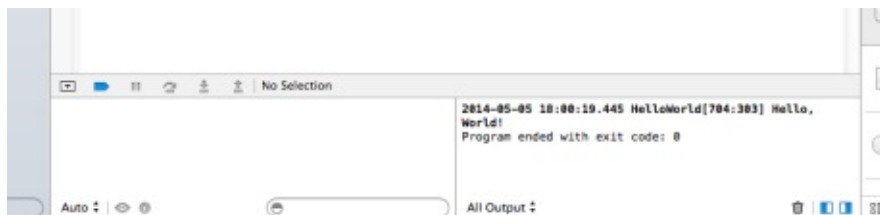
    }
    return 0;
}
```

@autoreleasepool ব্লকের মধ্যে আমরা আমাদের সবরকম প্র্যাকটিস কোড লিখবো। @autoreleasepool হচ্ছে মেমোরি লিক প্রতিহত করার একটা উপায়। যদিও এটা আমাদের টিউটোরিয়াল সিরিজের মূল বিষয় থেকে আলাদা, তবুও এই বিষয়ে জানতে পারেন [এখানে](#)। Apple ডকু থেকে - *"The @autoreleasepool statement is there to support memory management for your app. Automatic Reference Counting (ARC) makes memory management straightforward by getting the compiler to do the work of keeping track of who owns an object; @autoreleasepool is part of the memory management infrastructure."*

তো, @autoreleasepool এর কথা আপাতত মাথা থেকে বাদ দিলে বলা যায় যে, main() ফাংশন সিম্পলি NSLog() নামের একটা গ্লোবাল ফাংশনকে কল করে যেটা Foundation ফ্রেমওয়ার্কে এ ডিফাইন করা আছে। Xcode console এ ম্যাসেজ প্রিন্ট করার জন্য এটা ব্যবহার করা হয়। শুরুতে @ চিহ্ন ব্যবহার করে Objective-C string টাইপের ভ্যালু ডিফাইন করা হয়, যেটা পাঠানো হচ্ছে NSLog() এর প্যারামিটার হিসেবে।

Command+R চেপে অ্যাপটি রান করলে Xcode এর Editor এরিয়ার নিচের ডিবাগ কনসোলে নিচের মত



ম্যাসেজ দেখা যাবে।  এই টিউটোরিয়াল সিরিজে আমরা main.m ফাইল এডিট করে করে Objective-C ল্যঙ্গুয়েজ ফিচার টেস্ট করবো। কিন্তু বাস্তবে iOS/OSX অ্যাপ তৈরির সময় এই ফাইলে হাত দেয়ার দরকার পরবে না। কারণ, Xcode দিয়ে অন্য টাইপের অ্যাপ টেমপ্লেট তৈরি করলে দেখবেন main() ফাংশনের মধ্যে application object এবং app delegate ইন্সট্যান্স

তৈরি করে পুরো প্রোগ্রাম এর কন্ট্রোল ছেড়ে দেয়া হচ্ছে Application Delegate এর কাছে এবং এটা করেই main() এর কাজ শেষ। তো, ব্যাপার হচ্ছে, এটাও আপাতত গবেষণার বিষয় না। আপাতত আমরা ল্যঙ্গুয়েজ হিসেবে Objective-C এর ফিচার টেস্ট করবো, অ্যাপ ডেভেলপমেন্ট নয়।

পরের পোস্টে ব্যাসিক সিনট্যাক্স, ভ্যারিয়েবল, কন্ডিশন, লুপ, ম্যাক্রো, ডাটা স্ট্রাকচার নিয়ে আলোচনা হবে।

Originally Posted [Here](#)

Objective-C এর ভিতরে আমাদের প্রিয় C

ভূমিকাঃ টাইটেল দেখে যদি আপনার মনে এই ধারণা হয়ে থাকে যে, "Objective-C এর মধ্যে কি C ব্যবহার করা যাবে?" অথবা "Objective-C কি C এর উপড়ের লেয়ারের কিছু?" তাহলে আপনার দুটো ধারণাই একদম ঠিক :) আগেও বলা হয়েছে, Objective-C হচ্ছে ট্র্যাডিশনাল C কে অবজেক্ট ওরিয়েন্টেড ফিচার দিয়ে তৈরি হওয়া আরেকটি ল্যান্ডুয়েজ। C যা যা করতে পারে, Objective-C ও তাই তাই করতে পারে কিন্তু Objective-C এর কিছু কাজ হয়ত C কে দিয়ে হবে না। অর্থাৎ Objective-C কে C এর সুপারসেট বলতে পারেন। C এর সোর্স আপনি Objective-C এর সোর্স ফাইলের মধ্যেই একই সাথে লিখতে পারেন এবং কম্পাইলার এর কাছে পাঠাতে পারেন। আর তাই, Objective-C এর উপর দখল আনার জন্য আপনাকে সেই মাদার ল্যান্ডুয়েজ C এর ব্যাসিক জানতেই হচ্ছে। এই ব্যপারটাকে মাথায় রেখে এই চ্যাপ্টারে ব্যাসিক C এর ব্যপারগুলো ঝালাই করার চেষ্টা করা হবে এবং অবশ্যই তা আমাদের Objective-C এর সাথে কন্সাইন করেই। অর্থাৎ Objective-C এর ভিতরে আমাদের প্রিয় C

কমেন্টঃ কিছু বলার নাই। দেখেই বোঝা যাচ্ছে কিভাবে C কোডের মধ্যে কমেন্ট লেখা যায়।

```
// This is an inline comment

/* This is a block comment.
   It can span multiple lines. */
```

ভ্যারিয়েবলঃ একধরনের কন্টেইনার যা কিছু ভ্যালু ধারণ করতে পারে। C তে ভ্যারিয়েবল গুলো Typed অর্থাৎ ভ্যারিয়েবল ডিক্লেয়ার করার সময় জানাতে হবে এটা কি ধরনের ভ্যালু ধারণ করবে। ভ্যারিয়েবলকে `<Data Type>` `<Variable Name>` এই প্যাটার্নে ডিক্লেয়ার করা হয়। যেমন, আমাদের আগের চ্যাপ্টারে করা Command Line Tool টাইপের অ্যাপটির main.m ফাইল নিচের মত করে এডিট করে Command+R চেপে রান করে দেখতে পারি,

```
#import

int main(int argc, const char * argv[])
{

    @autoreleasepool {

        double bill = 100.50;
        NSLog(@"Total bill: %.2f", bill);

    }
    return 0;
}
```

এখানে bill একটি double টাইপ ভেরিয়েবল যেটা আমরা কনসোলে প্রিন্ট করছি।

কন্সট্যান্টঃ যে ভেরিয়েবলের ভ্যালু চেঞ্জ করা যায় না। কম্পাইলারকে সেই ভেরিয়েবলের কথা জানাতে const নামক ভেরিয়েবল মডিফায়ার ব্যবহার করা হয়।

```
#import

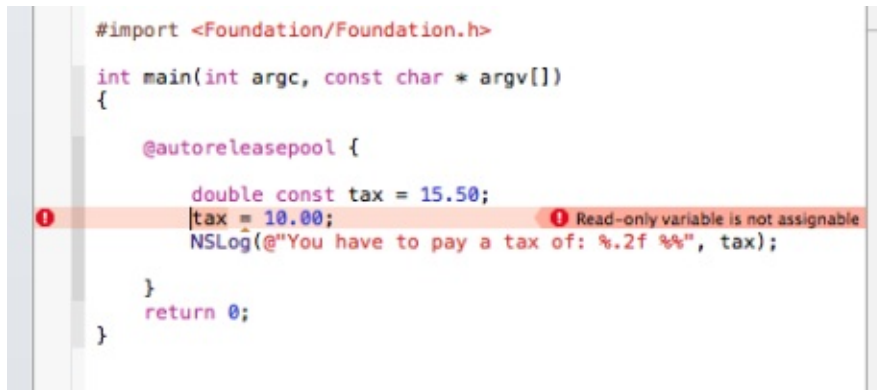
int main(int argc, const char * argv[])
{

    @autoreleasepool {

        double const tax = 15.50;
        tax = 10.00;
        NSLog(@"You have to pay a tax of: %.2f %%", tax);

    }
    return 0;
}
```

উপরের মত করে ড্যালাু চেঞ্জ করতে গেলে প্রথমেই Xcode বাধা দিবে নিচের মত করে। অন্য ভাবে কম্পাইল করতে গেলেও কম্পাইলার এরর দিবে। তাই, লাইন নম্বর 9 এখানে অবাস্তব।



অ্যারিদম্যাটিকঃ আমাদের আলোচিত অ্যাপ এর main.m ফাইল নিচের মত করে পরিবর্তিত করে Command+R চেপে রান করে দেখলেই বুঝতে পারবেন কোডের কমেন্ট এ দেয়া বর্ণনা অনুযায়ী C এর ব্যাসিক অ্যারিদম্যাটিক অপারেশন গুলো মনে পরছে কিনা।

```
#import

int main(int argc, const char * argv[])
{

    @autoreleasepool {

        NSLog(@"6 + 2 = %d", 6 + 2);    // Addition. Output: 8
        NSLog(@"6 - 2 = %d", 6 - 2);    // Subtraction. Output: 4
        NSLog(@"6 * 2 = %d", 6 * 2);    // Multiplication. Output: 12
        NSLog(@"6 / 2 = %d", 6 / 2);    // Division. Output: 3
        NSLog(@"6 %% 2 = %d", 6 % 2);    // % modulo operator for getting Remainder. Output: 0

        int i = 0;
        i++;    // increment operator
        NSLog(@"I was 0 and now I am: %d", i);

    }
    return 0;
}
```

কন্ডিশনালঃ C তে অন্যান্য ল্যাঙ্গুয়েজের মত if else ফিচার আছে যার ব্যবহার নিচের মত,

```
#import

int main(int argc, const char * argv[])
{

    @autoreleasepool {

        double bill = 500;
        if (bill >= 400) {
            NSLog(@"You ate enough");
        } else if (bill
```

এর সাথে সাথেই চলে আসে সবচেয়ে সাধারণ লজিক্যাল অপারেটর গুলোর কথা যেগুলো এই if else এর সাথে সব সময়ই ব্যবহৃত হয়,

```
a == b    Equal to
a != b    Not equal to
a > b     Greater than
a >= b    Greater than or equal to
a < b     Less than
a
```

আর switch তো আছেই। কিন্তু switch এ শুধু মাত্র integer ভেরিয়েবল ব্যবহার করা যায় এর সুইচিং ফ্যাক্টর হিসেবে।

```
#import

int main(int argc, const char * argv[])
{

    @autoreleasepool {

        int bill = 50;

        switch (bill) {
            case 500:
                NSLog(@"You ate exactly 500 Tk.");
                break;
            case 400:
                NSLog(@"You ate exactly 100 Tk");
                break;
            case 300:
            case 200:
                NSLog(@"You ate less than 300 Tk.");
                break;
            default:
                NSLog(@"Do not you how much you ate.");
                break;
        }

    }

    return 0;
}
```

উপরের if else এবং switch এর ব্যবহার ওয়ালা উদাহরণ দুইটা main.m ফাইলে লিখে শুধুমাত্র Command+R চেপেই রান করে দেখতে পারেন Xcode এর ডিবাগ কনসোলে কি আউটপুট আসে।

লুপঃ কোন ভ্যালুর মান ও অবস্থার উপর নির্ভর করে একটা কাজ বার বার করা হয় for এবং while লুপ ব্যবহার করে নিচের মত করে,

```
#import

int main(int argc, const char * argv[])
{

    @autoreleasepool {

        int totalItem = 5;

        for (int itemNumber = 1; itemNumber
```

```
#import

int main(int argc, const char * argv[])
{

    @autoreleasepool {

        int totalItem = 15;

        int itemNumber = 1;
        while (itemNumber <= totalItem) {
            if (itemNumber > 5) {
                NSLog(@"Please, Do not add any more food in my dish");
                break;
            }
            NSLog(@"Added item number: %d", itemNumber);
            itemNumber++;
        }

    }
    return 0;
}
```

ম্যাক্রোঃ সিঙ্গেলিক কম্প্যাঙ্ক ডিফাইন করার এক ধরনের লো-লেভেল এর পদ্ধতি। এটা ঠিক কম্প্যাঙ্ক ডেরিয়েবলের মত নয়। #define ডিরেক্টিভ ম্যাক্রো এর নাম থেকে Expansion ঠিক করে দেয়। মূলত Expansion হচ্ছে কিছু ক্যারেক্টারের সমন্বয়। কম্পাইলার কোড পড়ে ফেলার আগেই প্রিপ্রসেসর সব গুলো ম্যাক্রো এর নামের জায়গায় সেটার Expansion দিয়ে রিপ্লেস করে দেয়। নিচের উদাহরণটা দেখলে পরিষ্কার হবে আশা করছি,

```
#import

#define PI 3.14159

int main(int argc, const char * argv[])
{

    @autoreleasepool {

        NSLog(@"Whats the value of PI? It is: %f", PI);

    }
    return 0;
}
```

এখানে দেখুন কিভাবে main() ফাংশনের স্কোপের বাইরে থেকেও PI এর একটা মান ব্যবহার করা যাচ্ছে। এখানে PI কে অবজেক্ট টাইপ ম্যাক্রো বলা হয়। C তে ফাংশন টাইপ ম্যাক্রোও আছে যেমন নিচের উদাহরণ,

```
#import

#define PI 3.14159
#define circleName @"Pizza"
#define RAD_TO_CIRCUM(radius) (2*PI*radius) // Calculate circumference of a circle

int main(int argc, const char * argv[])
{

    @autoreleasepool {

        int radius = 6;
        NSLog(@"Circumference of this %@ is: %f", circleName, RAD_TO_CIRCUM(radius));

    }
    return 0;
}
```

এখানে RAD_TO_CIRCUM() একটি ফাংশন টাইপ ম্যাক্রো যেটা কিনা আবার আর্গুমেন্টও রিসিভ করতে পারে।

স্ট্রাকচারঃ C এর struct ব্যবহার অনেক গুলো বিভিন্ন রকম ভেরিয়েবলকে একত্রিত করে একটা নতুন ডাটা প্যাকেজ/গ্রুপ হিসেবে ডিফাইন করা যায়। পরে সেই ডাটা গ্রুপটাকে একটা অবজেক্ট এর মত ব্যবহারও করা যায়। নিচে একটা পিৎজার জন্য দরকারি তথ্যের ভেরিয়েবল গুলো গ্রুপ করে একটা নতুন ডাটা স্ট্রাকচার তৈরি করা হয়েছে। আর C এর typedef (নতুন data type ডিক্লেয়ার করার পদ্ধতি) ব্যবহার করে এই নতুন স্ট্রাকচারটিকে Pizza টাইপের data type হিসেবে যাতে ডিল করা যায় সে ব্যবস্থা করা হয়েছে।

```
#import

typedef struct {
    unsigned int diameter;
    char mainIngredient[10];
    char pizzaName[10];
} Pizza;

int main(int argc, const char * argv[])
{

    @autoreleasepool {

        Pizza makePizza = {6, "beef", "italian"};
        NSLog(@"You are making one : %d inch %s %s pizza!", makePizza.diameter, makePizza

    }
    return 0;
}
```

এখানে ফাংশনের কাজের শুরুতে makePizza নামের একটি Pizza টাইপ ডেরিয়েবল (এক্ষেত্রে স্ট্যাকচার) এর জন্য ডাটা পপুলেট/এসাইন করা হয়েছে একটি Initializer syntax এর মাধ্যমে।

ইনিউমারেশন (**enum**) :: enum কি-ওয়ার্ড ব্যবহার করে একটি enumerated type তৈরি করা হয় যেটাকে আসলে একাধিক কন্সট্যান্ট ডেরিয়েবলের (enumerators) সমষ্টিও বলা যেতে পারে যেখানে কন্সট্যান্ট ডেরিয়েবল গুলোর মান বাই ডিফল্ট সিরিয়ালি 0,1,2,3 ... হিসেবে ডিফাইন হয়ে থাকে।

```
#import

typedef enum {
    Beef,
    Chicken,
    Mutton
} Pizza;

int main(int argc, const char * argv[])
{
    @autoreleasepool {

        Pizza makePizza = Chicken;

        NSLog(@"Chicken has position %u in the enumerated type Pizza.", makePizza);

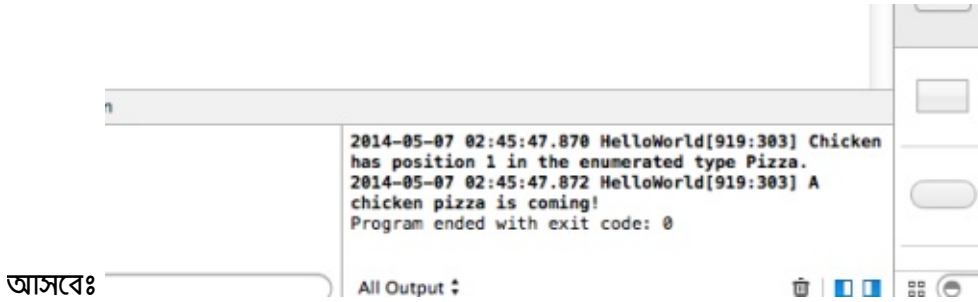
        switch (makePizza) {
            case Beef:
                NSLog(@"A beef pizza is coming!");
                break;
            case Chicken:
                NSLog(@"A chicken pizza is coming!");
                break;
            case Mutton:
                NSLog(@"A mutton pizza is coming!");
            default:
                break;
        }

    }

    return 0;
}
```

প্রথমত, আগের মত এই উদাহরণেও typedef ব্যবহার করা হয়েছে। অতঃপর, এখানে যে Enumerated type টি তৈরি করা হয়েছে তার মধ্যে Beef একটি কন্সট্যান্ট ডেরিয়েবল যার মান 0, Chicken এর 1 এবং Mutton এর 2 এভাবে কল্পনা করা যেতে পারে। আর ঠিক এভাবেই, switch এর যে একমাত্র integer টাইপের উপরই নির্ভরতা সেটা দূর করে এর মধ্যে প্রয়োজনে string নিয়েও কাজ করা যেতে পারে। এই প্রোগ্রামটি রান করলে নিচের মত আউটপুট

প্রথমত, আগের মত এই উদাহরণেও typedef ব্যবহার করা হয়েছে। অতঃপর, এখানে যে Enumerated type টি তৈরি করা হয়েছে তার মধ্যে Beef একটি কন্সট্যান্ট ভেরিয়েবল যার মান 0, Chicken এর 1 এবং Mutton এর 2 এভাবে কল্পনা করা যেতে পারে। আর ঠিক এভাবেই, switch এর যে একমাত্র integer টাইপের উপরই নির্ভরতা সেটা দূর করে এর মধ্যে প্রয়োজনে string নিয়েও কাজ করা যেতে পারে। এই প্রোগ্রামটি রান করলে নিচের মত আউটপুট



অ্যারেঃ যদিও iOS/OSX অ্যাপ ডেভেলপমেন্টের সময় Foundation ফ্রেমওয়ার্কের সাথে থাকা হাই লেভেল NSArray এবং NSMutableArray ক্লাস গুলো দিয়ে অ্যারে অপারেশন করাই হবে সবচেয়ে সুবিধা জনক তবুও এখানে যেহেতু C নিয়েই একটু স্মৃতিচারণ করা হচ্ছে এবং C এর অ্যারেকেও Objective-C এর সাথে ব্যবহার করা যাবে তাই নিচে থাকছে অ্যারে এর উদাহরণ,

```
#import

int main(int argc, const char * argv[])
{

    @autoreleasepool {

        int pizzaSize[3] = {6, 12, 18};

        for (int i=0; i
```

অ্যারে ডিক্লেয়ার করার int pizzaSize[3] স্টেটমেন্টটি শুরুতেই ৩টা অনুক্রমিক মেমোরি ব্লক অ্যালোকেট করে নেয় যেখানে integer টাইপের ডাটা অনায়াসে ধরবে। তারপরেই initializer syntax ব্যবহার করে সেই অ্যারেটিকে পপুলেট করা হচ্ছে অর্থাৎ ডাটা এসাইন করা হচ্ছে। এরপর pizzaSize[] এর মধ্যে i এর মান অর্থাৎ 0, 1 বা 2 অফসেট হিসেবে দিয়ে ওই অ্যারের ড্যালু গুলো এক্সেস করা হচ্ছে। যেহেতু অ্যারেতে যেহেতু অনুক্রমিক ভাবেই ডাটা থাকে তাই অফসেট এর সাথে ১ যোগ করে দিয়ে দিয়েই পরের অফসেটের ড্যালু পাওয়া যাচ্ছে।

পয়েন্টারঃ পয়েন্টার হচ্ছে মেমোরি অ্যাড্রেস এর রেফারেন্স। একদিকে যেমন, ভেরিয়েবল হচ্ছে ড্যালু রাখার কন্টেইনার অন্যদিকে পয়েন্টার হচ্ছে মেমোরিতে ড্যালুটি যেখানে জমা আছে সেখানকার অ্যাড্রেস এর রেফারেন্স হোল্ডার। & হচ্ছে রেফারেন্স অপারেটর যা একটি সাধারণ ভেরিয়েবলের মেমোরি অ্যাড্রেস রিটার্ন করে। সেটাই পয়েন্টার এর মধ্যে জমা রাখা হয়। আর যখন সেই মেমোরি অ্যাড্রেস এ থাকা ড্যালুটির দরকার হয় তখন * ডি-রেফারেন্স অপারেটর ব্যবহার করে পয়েন্টারে জমা থাকা মেমোরি অ্যাড্রেস এর কন্টেন্ট বা ড্যালুকে এক্সেস করা যায়। আবার এই ডি-রেফারেন্স অপারেটর ব্যবহার করেই ওই মেমোরি অ্যাড্রেসে নতুন ড্যালু সেটও করা যায়। নিচের উদাহরণ দেখলে আরেকটু ক্লিয়ার হবে,


```
#import

int main(int argc, const char * argv[])
{

    @autoreleasepool {
        // Define a normal variable
        int bill = 500;
        // Declare a pointer that points to an int
        int *pointer;

        // Find the memory address of the variable
        pointer = &bill;
        // Dereference the address to get its value
        NSLog(@"Your bill is: %d Tk", *pointer);

        // Assign a new value to the memory address
        *pointer = 525;
        // Access the changed value via the variable
        NSLog(@"Your bill including tax is: %d Tk", bill);

    }
    return 0;
}
```

Void Pointer নামের একধরনের পয়েন্টার আছে যা আসলে যেকোনো কিছুকেই পয়েন্ট করতে পারে। অর্থাৎ নির্দিষ্ট করে প্রথম থেকেই যে একটা integer বা char কে পয়েন্ট করবে/করছে, তা নয়। কিন্তু ওই ভয়েড পয়েন্টারে জমা থাকা মেমোরি অ্যাড্রেস ধরে সেখানকার ভ্যালু বা কন্টেন্ট এক্সেস করতে হলে একটু কাজ করতে হবে অর্থাৎ পয়েন্টার কে বলতে হবে ওই অ্যাড্রেস থেকে কোন ফরম্যাট হিসেবে ডাটা পড়বে। নিচে একটি উদাহরণ আছে,

```
#import

int main(int argc, const char * argv[])
{

    @autoreleasepool {

        int toPay = 500;

        void *genericPointer = &toPay;
        int *intPointer = (int *)genericPointer; // Casting to non-void pointer

        NSLog(@"To pay actually: %d", *intPointer);

    }
    return 0;
}
```

এখানে ভয়েড পয়েন্টারটিকে cast করে নন-ভয়েড এবং integer টাইপের পয়েন্টারে রূপান্তর করা হচ্ছে। (int *) স্টেটমেন্টটি ভয়েড পয়েন্টারের কন্টেন্ট কে integer হিসেবে interpret করছে।

Objective-C তে পয়েন্টারঃ Objective-C তে পয়েন্টারের ব্যাপারটা খুব কম কথাতেই শেষ হয়ে যাবে কিন্তু ভালো মতই মনে রাখতে হবে যে সব রকম Objective-C এর অবজেক্টকে পয়েন্টার হিসেবে উল্লেখ করা হয়। যেমন নিচের মত করে, একটা NSString অবজেক্টকে অবশ্যই পয়েন্টার হিসেবে স্টোর করতে হবে, নরমাল ডেরিয়েবল হিসেবে নয়।

```
NSString *foodItem = @"Beef Pizza";
```

ডেরিয়েবল ডিক্লেয়ারেশন বাদে বাকি সব সময়ের জন্য Objective-C এর সিনট্যাক্সকে পয়েন্টার হিসেবে কাজ করার উপযোগী করেই তৈরি করা হয়েছে। তাই, একবার একটি অবজেক্ট পয়েন্টার ডিফাইন করার পর থেকেই সেটাকে একটা নরমাল ডেরিয়েবল মনে করে সেটার সাথে ডিল করা যেতে পারে। এতে করে ব্যাপারটা সহজবোধ্য হবে।

পরের চ্যাপ্টারঃ ব্যাসিক C এর ফাংশন নিয়ে একটু আলোচনা করেই Objective-C তে ফাংশনের ডিক্লেয়ারেশন, ডেফিনেশন নিয়ে আলোচনা হবে। তার পর পরই, এই যে Objective-C আমাদের প্রিয় C এর সাথে object oriented feature জুড়ে দিয়ে C দিয়েও OOP বেজড প্রোগ্রামিং করার রাস্তা তৈরি করলো, সেটার ব্যবহার অর্থাৎ Objective-C এর ক্লাস, প্রোপার্টি, মেথড নিয়ে আলোচনা হবে।

Originally Posted [Here](#)

ফাংশন ও এর ব্যবহার

ভূমিকাঃ গত অধ্যায়ে Objective-C এর সাথে C এর সম্পর্ক এবং C তে কমেন্ট, ভ্যারিয়েবল, কন্সট্যান্ট, অ্যারিদম্যাটিক অপারেশন, কন্ডিশনাল অপারেশন, লুপ, ম্যাক্রো, স্ট্রাকচার, ইনিওমারেশন, অ্যারে, পয়েন্টার ইত্যাদির ব্যবহার শিখেছি যা C এবং Objective-C তথা সকল প্রোগ্রামিং ল্যাঙ্গুয়েজের অতি সাধারণ কিছু ফিচার। ফাংশনও এদের মতই আর একটি বিষয় যা সকল আধুনিক প্রোগ্রামিং ল্যাঙ্গুয়েজের একটি সাধারণ ফিচার বা কম্পোনেন্ট।

কি এবং কেন? ফাংশন নির্দিষ্ট কোড ব্লকে সম্পূর্ণ অ্যাপ্লিকেশনে রিইউজ করার সুবিধা দেয়। ধরি, আমি খুব ভোজন রসিক এবং নিজেই নতুন নতুন রান্না করে খেতেই বেশি পছন্দ করি। একারণে আমি একেকবার একেক জনের কাছ থেকে বিভিন্ন খাবারের রেসিপি জেনে নেই। এবার ধরি, আমি একজন নির্দিষ্ট বাবুর্চি চিনে রাখলাম যার কাজ হবে আমাকে রেসিপি জানানো। সে যেখানেই থাকুক না কেন আমি তাকে ফোন করে খাবারের নাম বলব এবং সে আমাকে রেসিপিটা জানাবে। এখন কিন্তু একেক বার একেক লোকের সাহায্য লাগবে না আমার। একজনই থাকবে, আমার যখন দরকার পাবে, তখন সেই একজনকেই আমি ব্যবহার করব। এই নির্দিষ্ট একজন লোকই প্রোগ্রামিং এর ভাষায় ফাংশন। আমরা আমাদের সাধারণ এবং সবসময় যেগুলো লাগে সেই কাজ গুলোকে কিছু স্বাধীন ভাগে ভাগ করব এবং সেই একেক ভাগকে আলাদা আলাদা ফাংশন হিসেবে ডিফাইন (কার্যকলাপ নির্ধারণ করে দেয়া) করব। তাহলে একই কাজের জন্য বার বার কোড লিখতে হবে না। শুধু ওই ফাংশনকে ডাকলেই (ফাংশন কল) হবে।

ফাংশনের ব্যাসিক সিন্ট্যাক্সঃ একটি ফাংশনের মোট চারটি অংশ থাকে-

- রিটার্ন ভ্যালু (এই ফাংশনটি কাজ শেষে যদি কোন কিছু ফেরত/ফলাফল দেয় তাহলে সেটির ডাটাইপ কি হবে)
- নাম (ফাংশনটির একটি নির্দিষ্ট নাম থাকবে যা দিয়ে তাকে ডাকা হবে বার বার)
- প্যারামিটারস (ফাংশনটির কাজ সম্পূর্ণ করতে যদি তথ্যের দরকার পড়ে তাহলে সেই তথ্য গুলো)
- কোড ব্লক (এই অংশে ফাংশনটা যে কাজ করবে তার প্রোগ্রাম্যাটিক্যাল কোড/লজিক থাকে)

উদাহরন স্বরূপ নিচের কোডটুকু দেখা যাক,

```
#import

void billKotoHolo()
{
    NSLog(@"স্বাগত আপনার বিন হয়েছে ১২০০ টাকা");
}

int main(int argc, const char * argv[])
{
    @autoreleasepool {
        billKotoHolo();
    }
    return 0;
}
```

এখানে দেখা যাচ্ছে billKotoHolo() একটা ফাংশন। মেইন main() ফাংশন এর মধ্যে থেকে একে কল করা হয়েছে। এই ফাংশনটি মেইন ফাংশনকে কোন কিছু রিটার্ন করবে না তাই এর রিটার্ন টাইপ void। এই ফাংশনের কার্যকলাপ বলতে গেলে সিমপ্লি কনসোলে একটা ম্যাসেজ প্রিন্ট করা আর এই কাজটুকু করতে এর কোন প্যারামিটার দরকার নাই। তাই ডেফিনেশন এর সময় () এর মধ্যে কিছু নাই। {} এর ভিতরের অংশই কোডব্লক যেখানে বলা আছে এই ফাংশনের কাজ কি। আবার, কল করার সময়ও কোন প্যারামিটার পাঠানো হয়নি তাই () ব্র্যাকেটের ভিতরের অংশ ফাকা। অর্থাৎ কোন প্যারামিটার পাস করা হচ্ছে না।

ফাংশনে, পয়েন্টার রেফারেন্সকেও রিটার্ন ড্যালা এবং প্যারামিটার হিসেবে ব্যবহার করা যায়। অর্থাৎ ফাংশনগুলো খুব সহজেই Objective-C এর অবজেক্ট গুলোর সাথে ব্যবহৃত হতে পারে। কারন, আগেই বলা হয়েছে যে, "Objective-C তে সব ধরনের অবজেক্টকে পয়েন্টার হিসেবে প্রকাশ করা হয়/যায়।" এখন নিচের উদাহরণটা খেয়াল করুন,

```
#import

int getCount(NSArray *foods) {
    int count = (int)[foods count];
    return count;
}

int main(int argc, const char * argv[])
{

    @autoreleasepool {
        NSArray *foods = @[@"Chicken Fry", @"Soup", @"Biriani", @"Snacks"];
        NSLog(@"%d", getCount(foods));    }
    return 0;
}
```

getCount() ফাংশনটি আর্গুমেন্ট হিসেবে NSArray অবজেক্ট (Objective-C তে একটি অ্যারেটাইপ ড্যারিয়েবল যাতে আমরা রেইসুরেক্টের খাবারের লিস্ট রাখছি) নেয় এবং integer টাইপ ডাটা (মোট আইটেমের সংখ্যা) রিটার্ন করে।

```
int count = (int)[foods count];
```

এই লাইন টি দেখে ঘাবড়ানোর কিছু নেই। foods অ্যারেতে কতগুলো ইলিমেন্ট আছে সে সংখ্যাটি count রিটার্ন করে। count হচ্ছে food অবজেক্টের একটি বিল্ট-ইন মেথড যেটা আসলে NSArray ক্লাসে ডিফাইন করা আছে। পরবর্তীতে বিস্তারিত দেখানো হবে।

ডিক্লেয়ারেশন বনাম ইম্প্লিমেন্টেশনঃ যেখানে একটি ফাংশনকে কল করে ব্যবহার করা হবে তার পূর্বেই কোথাও ফাংশনটিকে ডিফাইন করতে হয়। অর্থাৎ ব্যবহারের পূর্বেই কম্পাইলার কে জানাতে হবে যে getCount() নামে একটি ফাংশন আছে এবং এই ফাংশনটির কাজের বর্ণনা তথা ডেফিনিশন/ইম্প্লিমেন্টেশনও ঠিক করা আছে। উপরের কোড এ getCount() ফাংশনটি main() এর আগে না লিখে পরে লিখে main() এর ভিতর থেকে কল করলে কম্পাইলার ফাংশনটিকে খুজে পেত না।

ফাংশনের ডিক্লেয়ারেশন কম্পাইলারকে ফাংশনটির ইনপুট, আউটপুট সম্পর্কে জানায়। শুধুমাত্র রিটার্ন ডাটার ডাটাইপ ও প্যারামিটারস পাঠিয়ে কম্পাইলার চেক করতে পারে যে ফাংশনটির ব্যবহার ঠিকমত হচ্ছে কিনা। উদাহরণস্বরূপ বলা যায়, ডিক্লেয়ারেশন এ রিটার্ন টাইপ হিসেবে ঠিক করা আছে int কিন্তু আসলে কোডব্লক এ ডাটা

পাঠাচ্ছে double, তাহলে কম্পাইলার এই ভুলটা ধরতে পারে। এই ডিক্লেয়ারেশন এর সাথে একটি কোডব্লক (ইমপ্লিমেন্টেশন) এটাচ করলেই তা হয়ে যায় সম্পূর্ণ ডেফিনিশন। নিচের কোডটি দেখলেই ব্যাপারটি পরিষ্কার হয়ে যাবে,

```
// main.m
#import

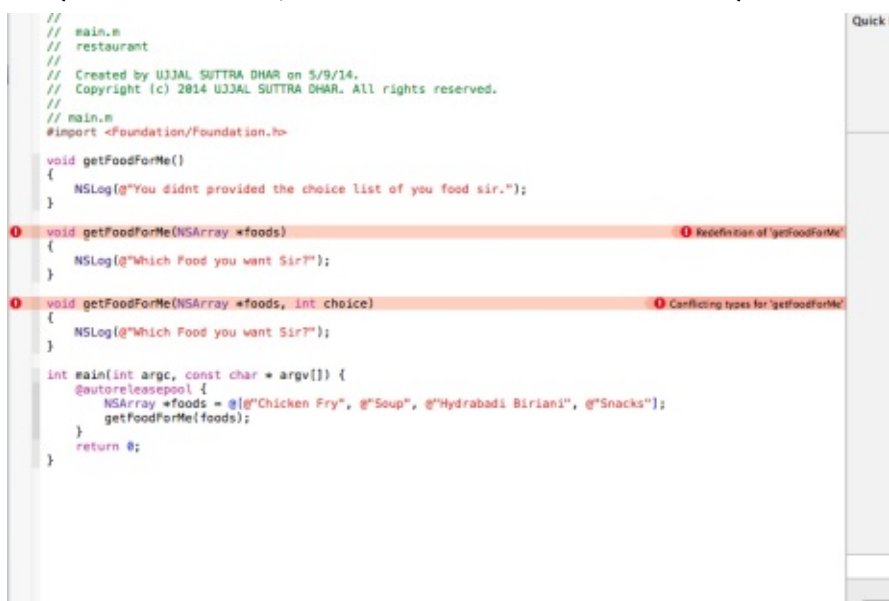
// Declaration
int getCount(NSArray *);

int main(int argc, const char * argv[]) {
    @autoreleasepool {
        NSArray *foods = @[@"Chicken Fry", @"Soup", @"Hydrabadi Biriani", @"Snacks"];
        NSLog(@"%d", getCount(foods));
    }
    return 0;
}

// Implementation
int getCount(NSArray *foods) {
    int count = (int)[foods count];
    return count;
}
```

এখানে আগে ফাংশনটিকে ডিক্লেয়ার করা হয়েছে এবং এটার ব্যাপারে কম্পাইলারকে জানানো হচ্ছে এবং পরে যেকোনো এক জায়গায় সেটার পূর্ণ ইমপ্লিমেন্টেশন লেখা হয়েছে। উল্লেখ্য, রিটার্ন টাইপে সকল ধরনের বেসিক ডাটাটাইপ, পয়েন্টার ও অবজেক্ট হতে পারে। কিন্তু খেয়াল রাখতে হবে যে, ফাংশন শুধু মাত্র একটি ইনফরমেশনই রিটার্ন করতে পারে। হতে পারে সেটি একটি অবজেক্ট অথবা একটি অ্যারে অথবা একটি পয়েন্টার অথবা অন্য যেকোন ডাটাটাইপ।

ফাংশন ওভারলোডিংঃ C এর মত Objective-C তেও ফাংশন ওভারলোডিং নাই। আসলে অবজেক্ট ওরিয়েন্টেড প্রোগ্রামিং এর প্রধান যে ৩ টি ফিচার আছে তার মধ্যে একটি হল পলিমরফিজম। ফাংশন ওভারলোডিং পলিমরফিজম এরই উদাহরণ। পরবর্তীতে অধ্যায় গুলো এ সম্পর্কে বিস্তারিত আলোচনা হবে। নিচের উদাহরণটি



```
//
//  main.m
//  restaurant
//
//  Created by UJJAL SUTTRA DHAR on 5/9/14.
//  Copyright (c) 2014 UJJAL SUTTRA DHAR. All rights reserved.
//
//  main.m
#import <Foundation/Foundation.h>

void getFoodForMe()
{
    NSLog(@"You didnt provided the choice list of you food sir.");
}

void getFoodForMe(NSArray *foods)
{
    NSLog(@"Which Food you want Sir?");
}

void getFoodForMe(NSArray *foods, int choice)
{
    NSLog(@"Which Food you want Sir?");
}

int main(int argc, const char * argv[]) {
    @autoreleasepool {
        NSArray *foods = @[@"Chicken Fry", @"Soup", @"Hydrabadi Biriani", @"Snacks"];
        getFoodForMe(foods);
    }
    return 0;
}
```

দেখলেই বুঝা যাবে।

Static কি-ওয়ার্ড ঃ Static কি-ওয়ার্ডটি ফাংশন অথবা ড্যারিয়েবল এর অ্যাভেইলাবিলিটি (Availability) অস্টার করার সুবিধা দেয়। মূলত এই কিওয়ার্ডের এফেক্ট নির্ভর করে কোন ক্ষেত্রে এর ব্যবহার হচ্ছে তার ওপর। Static কিওয়ার্ডটি ফাংশন অথবা ড্যারিয়েবল ডিক্লারেশন এ ব্যবহার করা হয়। এই অংশে স্ট্যাটিক কিওয়ার্ডের ব্যবহার দেখানো হবে।

স্ট্যাটিক ফাংশন (Static Function) ঃ স্বাভাবিকভাবেই সকল ফাংশনই গ্লোবাল স্কোপে থাকে। এ কথার মানে হল কোন ফাইলে ফাংশন ডিফাইন করার সাথে সাথে তা অন্য সকল ফাইল থেকে কল করা যায় যদি ওই ফাংশন ওয়ালা ফাইলটি কলার (caller) ফাইলে ইনক্লুড করে নেয়া হয়। ধরে নেই আমাদের প্রজেক্টে ৩ টা ফাইল আছে। কোন একটি ফাইলে যদি কোন ফাংশনের ডেফিনেশন থাকে তাহলে অন্য সকল ফাইল থেকে এই ফাংশন কে ব্যবহার করা যাবে। যদি আমরা চাই যে এই ফাংশনটি শুধুমাত্র এই ফাইল থেকেই ব্যবহার করা যাবে, অর্থাৎ এই ফাংশনটি এই ফাইলের প্রাইভেট ফাংশন হবে তাহলে ফাংশনটিকে স্ট্যাটিক হিসেবে ডিক্লেয়ার করলেই কাজটি হয়ে যাবে। বিভিন্ন ফাইলে যদি একই নামের ফাংশন থাকে তারপর ও কনফ্লিক্ট করবে না।

স্ট্যাটিক ফাংশনের বেসিক সিনট্যাক্স নিচের উদাহরণে দেখানো হল। এই কোডটুকু যদি অন্য কোন ফাইলে লিখা হয় যেমন কোন ফাংশন লাইব্রেরীতে, তাহলে কখনোই main.m এর মধ্যে থেকে getMenu() ফাংশনটিকে এক্সেস/কল করা যাবে না।

```
// Static function declaration
static NSArray getMenu();

// Static function implementation
static int getMenu() {
    NSArray *menuList = @[@"Chicken Fry", @"Soup", @"Biriani", @"Snacks"];
    return (int)[menuList count];
}
```

বিশেষভাবে উল্লেখ্য যে, static কিওয়ার্ডটি ডিক্লারেশন এবং ইমপ্লিমেন্টেশন উভয় ক্ষেত্রেই ব্যবহার করতে হবে।

স্ট্যাটিক লোকাল ড্যারিয়েবল: কোন ফাংশনের ভেতরে ডিক্লেয়ার করা ড্যারিয়েবল কে ওই ফাংশনের লোকাল ড্যারিয়েবল বলা হয়। অর্থাৎ ড্যারিয়েবলটির স্কোপ হবে ওই ফাংশনের ভেতরের অংশটুকুই। যতবার এই ফাংশনটি কল করা হবে ততবারই ড্যারিয়েবলটি নতুন করে রি-সেট হবে। "নতুন করে রি-সেট" হওয়া বলতে বোঝানো হচ্ছে যে, আগের বার কল হওয়ার সময় ড্যারিয়েবলটির আপডেটেড ভ্যালু, মেমোরি লোকেশন পরেরবারের সাথে মিলবে না। প্রত্যেক বার ফাংশনটি কল হবার সময় ড্যারিয়েবলটির নতুন করে মেমোরি লোকেশন সেট হবে এবং সেটির প্রাথমিক ইনিসিয়ালাইজড মান সেট হবে, তথা রিসেট হবে।

```
#import

int getNumberOfChicken()
{
    int count = 0;
    count = count + 1;
    return count;
}

int main(int argc, const char * argv[]) {
    @autoreleasepool {
        NSLog( @"%", getNumberOfChicken()); // 1
        NSLog( @"%", getNumberOfChicken()); // 1
    }
    return 0;
}
```

উপরের কোডে getNumberOfChicken() ফাংশনটিতে count ভ্যারিয়েবলটি স্ট্যাটিক না। তাই যতবারই main() থেকে getNumberOfChicken() কে কল করা হবে, ততবারই নতুন করে count ডিক্লেয়ার হবে (নতুন মেমরী লোকেশন এবং ভ্যালু ০)।

কিন্তু নিচের মত করে, এই ভ্যারিয়েবলটি ডিক্লেয়ার করার সময় যদি স্ট্যাটিক হিসেবে ডিক্লেয়ার করা হয় তাহলে প্রত্যেকবার কল হওয়ার সময় নতুন করে রি-সেট হবে না। আগের বার ফাংশনটি এক্সিকিউট হওয়ার পর ভ্যারিয়েবলটির যা ভ্যালু ছিল তাই থাকবে কারন মেমরী লোকেশন আগেরটাই থাকবে।

```
#import

int getNumberOfChicken()
{
    static int count = 0;
    count = count + 1;
    return count;
}

int main(int argc, const char * argv[]) {
    @autoreleasepool {
        NSLog( @"%", getNumberOfChicken()); // 1
        NSLog( @"%", getNumberOfChicken()); // 2
    }
    return 0;
}
```

ভ্যারিয়েবলটি static হিসেবে ডিক্লেয়ার করার কারনে শুধুমাত্র প্রথমবার count ভ্যারিয়েবলের জন্য মেমরী লোকেশন সেট হয় এবং ০ দিয়ে ইনিশিয়ালাইজ হয়। কিন্তু পরবর্তীতে যতবারই কল করা হোক না কেন count এর ভ্যালু ওই মেমরী লোকেশন থেকেই ইনডোক করা হয়।

ফাংশনের স্ট্যাটিক লোকাল ভ্যারিয়েবল গুলো যতবারই কল করা হোক না কেন, সাধারণ লোকাল ভ্যারিয়েবলের মত বার বার ওই ভ্যারিয়েবল রিসেট হয় না। ফাংশন ওই ভ্যারিয়েবল শেষ অবস্থার ডাটা মনে রাখে। কিন্তু স্ট্যাটিক ফাংশনের মত স্ট্যাটিক লোকাল ভ্যারিয়েবলের স্কোপের কোন পরিবর্তন হয় না। স্ট্যাটিক লোকাল ভ্যারিয়েবল শুধু মাত্র ওই ফাংশনের স্কোপেই ব্যবহার করা যাবে।

ফাংশন লাইব্রেরীঃ অবজেক্টিভ-সি নেমস্পেস সাপোর্ট করে না। অন্যান্য গ্লোবাল ফাংশন গুলোর সাথে যাতে নাম নিয়ে কোন কনফ্লিক্ট না হয় সে জন্য ফাংশন এবং ক্লাসগুলোতে ইউনিক আইডেন্টিফায়ার হিসেবে প্রিফিক্স যোগ করে দেওয়া হয়। ফলস্বরূপ শুধু `CaptureSession` এর পরিবর্তে `AVCaptureSession()`, শুধু `Layer` এর পরিবর্তে `CALayer()` ইত্যাদি ফাংশন দেখা যায়।

তাই আমরা যখন নিজেদের লাইব্রেরী বানাবো তখন ফাংশনের নাম গুলো একটা হেডার (.h) ফাইলে ডিক্লেয়ার করব এবং আলাদা ইমপ্লিমেন্টেশন (.m) ফাইলে ফাংশনগুলোর ইমপ্লিমেন্টেশন করব। Objective-C এর ক্লাস গুলোর গঠনও এরকম। এই পদ্ধতিতে ফাইল দুটোকে যথাক্রমে ইন্টারফেস ফাইল এবং ইমপ্লিমেন্টেশন ফাইল বলা হয়। এতে করে ওই লাইব্রেরী ব্যবহার করার সময় ইমপ্লিমেন্টেশন না ভেবে শুধুমাত্র হেডার ফাইলকে ইমপোর্ট করেই কাজ করা যাবে। উল্লেখ্য, হেডার ফাইলের এই ধরনের ব্যবহারকে একটা API লেয়ার হিসেবেও কল্পনা করা যায়। অর্থাৎ আমরা আরেকটা ক্লাসের সাথে যোগাযোগের জন্য সেটার এক্সেস-অ্যাবল লেয়ার তথা ইন্টারফেস ফাইলের সাথে যোগাযোগ করবো, মূল ইমপ্লিমেন্টেশন ফাইলের সাথে নয়।

```
// RestaurantUtilities.h
#import

NSString *RUgetRestaurantTitle();
int RUgetCountOfItems();
```

এই হেডার ফাইলের ইমপ্লিমেন্টেশন ফাইলটি নিচের মত হতে পারে,

```
// RestaurantUtilities.m
#import "RestaurantUtilities.h"

NSString *RUgetRestaurantTitle()
{
    return @"ভূতের ঝন্ডা";
}

int RUgetCountOfItems()
{
    NSArray *items = @[@"স্রষে ইনিশ", @"খিচুরী"];
    return (int)[items count];
}
```

এখন main.m এ RestaurantUtilities.h হেডার ফাইল কে ইমপোর্ট করে সকল ফাংশনগুলোকে ব্যবহার করতে পারবো। এখানে উল্লেখযোগ্য যে RestaurantUtilities.m এ যদি কোন static ফাংশনের ডিক্লোরেশন এবং ইমপ্লিমেন্টেশন থেকে থাকত তাহলে কিন্তু main.m থেকে সেই ফাংশনগুলো ব্যবহার করতে চাইলে কম্পাইলার এরর হবে, যার ব্যাখ্যা আগেই দেয়া হয়েছে।


```
// main.m
#import
#import "RestaurantUtilities.h"

int main(int argc, const char * argv[]) {
    @autoreleasepool {

        NSLog( getRestaurantTitle());    // ডুতের আছা
        NSLog( @"%d", getCountOfItems()); // 2

    }
    return 0;
}
```

পরের চ্যাপ্টারঃ এই চ্যাপ্টারে C এবং Objective-C তে ফাংশনের ব্যবহার, সাথে এদের এবং ভেরিয়েবলের স্কোপ, ডিক্লেয়ারেশন, ইমপ্লেমেন্টেশন, ইন্টারফেস/হেডার ফাইল, ইমপ্লেমেন্টেশন ফাইল ইত্যাদি বিষয় নিয়ে আলোচনা হয়েছে। পরবর্তী চ্যাপ্টারে আমরা Objective-C এর অবজেক্ট ওরিয়েন্টেড ফিচার নিয়ে বিস্তারিত আলোচনা করবো। সেখানে থাকবে কিভাবে ক্লাস ডিফাইন করা যায়, কিভাবে অবজেক্ট ইন্সট্যান্সিয়েট করা যায়, কিভাবে প্রোপার্টি সেট করা যায় এবং কিভাবে মেথড কল করা যায় ইত্যাদি।

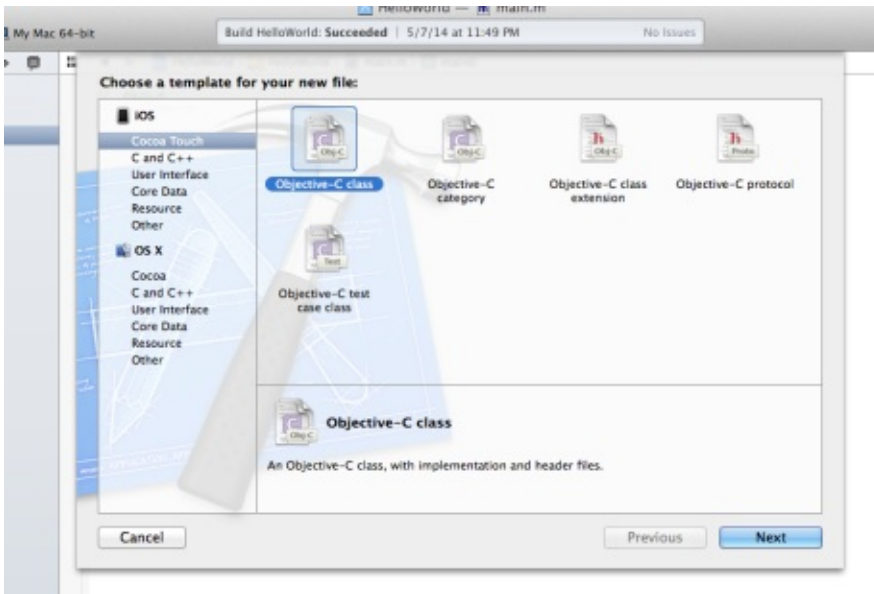
Originally Posted [Here](#)

ক্লাস ও অবজেক্ট

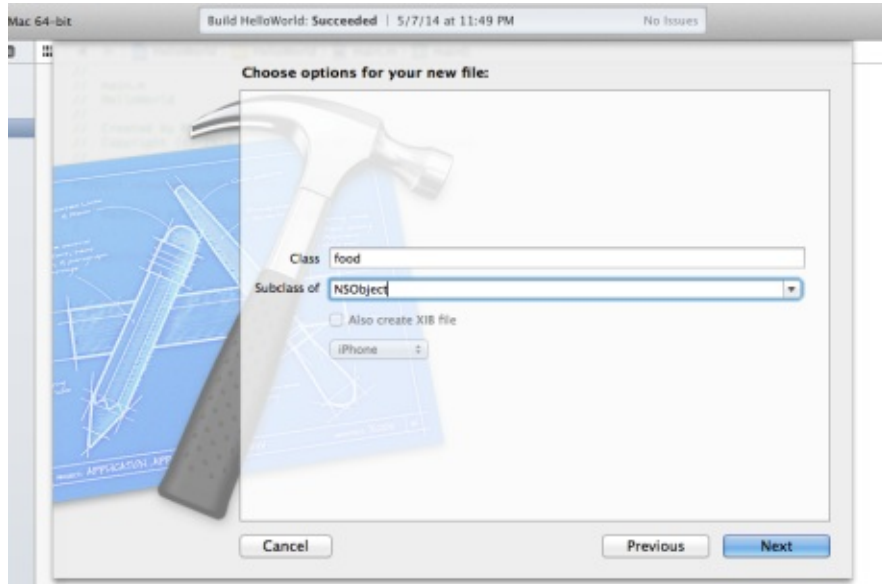
ভূমিকাঃ অন্যান্য অবজেক্ট ওরিয়েন্টেড প্রোগ্রামিং ল্যাঙ্গুয়েজের মতই অবজেক্টিভ-সি তেও - প্রোপার্টি, মেথড, ক্লাস, অবজেক্ট এই ব্যাপার গুলো বেশ ভালো ভাবেই আছে। আর এর আগে অনেক বার বলাই হয়েছে যে, সাধারণ সি ল্যাঙ্গুয়েজের সাথে অবজেক্ট ওরিয়েন্টেড ফিচার যুক্ত করেই অবজেক্টিভ-সি আত্মপ্রকাশ করেছে। এই অবজেক্ট ওরিয়েন্টেড ফিচারই মূলত সি থেকে অবজেক্টিভে-সি কে আলাদা করে। এখানেও একটি ক্লাসের মধ্যে কিছু রি-ইউজ্যাবল প্রোপার্টি-মেথড ডিফাইন করা এবং সেই ক্লাস থেকে একটি অবজেক্ট ইন্সট্যান্সিয়েট করে ওই প্রোপার্টি এবং মেথড গুলোর সাথে যোগাযোগ করার ব্যাপার গুলোও আছে। আর অবজেক্টিভে সি কে সি++ এর সাথে তুলনা করা যায় একটি জায়গায় সেটা হল, এটা একটা ক্লাসের ইন্টারফেসকে তার ইমপ্লিমেন্টেশন থেকে পৃথক করে। ইন্টারফেস ডিফাইন করে একটি ক্লাসের পাবলিক প্রোপার্টি ও মেথড গুলোকে আর ইমপ্লিমেন্টেশন ডিফাইন করে সেগুলোর মূল কার্যক্রম বা কার্যনীতি। আগের চ্যাপ্টারের ফাংশন এর ডিক্লেয়ারেশন + ইমপ্লিমেন্টেশন = ডেফিনেশন এর মতই।

ক্লাস তৈরিঃ এই চ্যাপ্টারে আমরা আদর্শ একটি ক্লাসের উদাহরণ হিসেবে food নামের একটি ক্লাস তৈরি করব যার ইন্টারফেস থাকবে food.h ফাইলে যেটাকে হেডারও বলা হয় আর ইমপ্লিমেন্টেশন থাকবে food.m ফাইলে। এ দুটোই Objective C ক্লাসের স্ট্যান্ডার্ড এক্সটেনশন। হেডার ফাইলের মাধ্যমে অন্য ক্লাস এই ক্লাসের সাথে যোগাযোগ করবে আর ইমপ্লিমেন্টেশন ফাইলটি মূলত কম্পাইলার এর কাছে প্রসেস হবে।

Xcode লঞ্চ করে আমাদের সেই Hello World নামক কমান্ড-লাইন টাইপ অ্যাপটাই ওপেন করুন। File -> New -> File এ গিয়ে নিচের মত করে Objective-C class সিলেক্ট করে Next করুন,

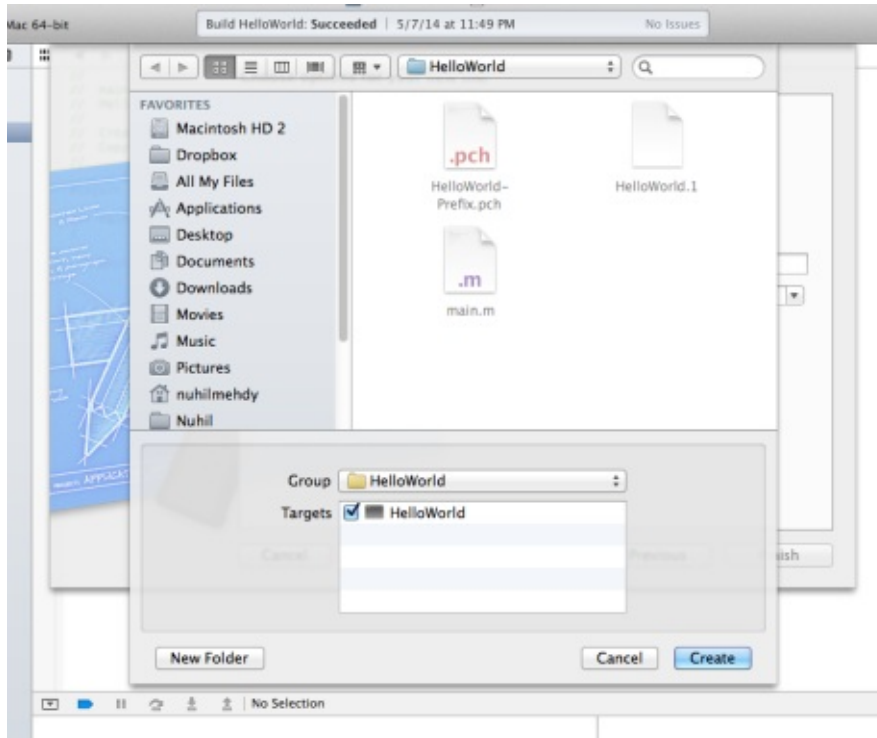


এরপর ক্লাসের নাম হিসেবে দিন food এবং Subclass of ফিল্ডে লিখুন NSObject. নিচের ছবির মত করে। তারপর Next ক্লিক করুন। NSObject হচ্ছে অবজেক্টিভ সি এর একটি টপ লেভেল ক্লাস যাকে মোটা মুঠি অন্য সবাই

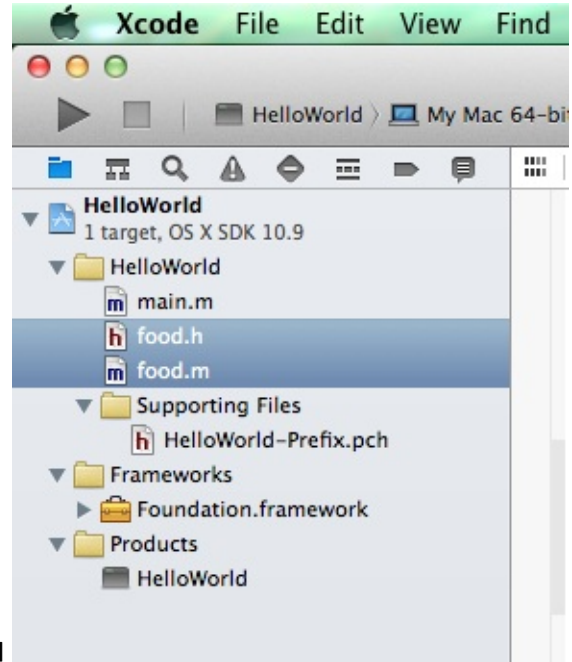


ইনহেরিট করে।

শেষে Group এবং Targets ঠিক মত সিলেক্ট/চেক করা আছে কিনা দেখে নিন এবং Create বাটনে ক্লিক করুন।



এখন আপনার প্রজেক্ট ন্যাভিগেটরে food.h এবং food.m নামের দুইটি নতুন ফাইল দেখতে পাবেন। এ দুটোই হচ্ছে



আমাদের food ক্লাসের ইন্টারফেস এবং ইমপ্লিমেন্টেশন ফাইল।

ইন্টারফেসঃ food.h ফাইলে ক্লিক করলেই ডান পাশের এডিটরে এই ফাইলের টেমপ্লেট কোড দেখতে পাবেন। আমরা এখানে একটি প্রোপার্টি এবং একটি মেথড ডিক্লেয়ার করবো। এডিট করার পর এই ফাইলের চেহারা হবে নিচের মত,

```
// food.h

#import

@interface food : NSObject

@property (copy) NSString *item;

-(void)grab;

@end
```

@interface ডিরেক্টিভ ব্যবহার করে ইন্টারফেস তৈরি করা হয় এবং এর পরেই ক্লাসের নাম এবং তার সুপার ক্লাসের নাম একটি কোলন দিয়ে পৃথক করে লেখা হয়। @property ডিরেক্টিভ ব্যবহার করে পাবলিক প্রোপার্টি ডিক্লেয়ার করা হয়। copy অ্যাট্রিবিউট দিয়ে এটির মেমোরি ম্যানেজমেন্ট এর ধরন বোঝানো হচ্ছে। item এ অ্যাসাইন করা ড্যালাু সরাসরি পয়েন্টার টাইপ না হয়ে কপি টাইপ হবে। সামনের চ্যাপ্টারে এ ব্যাপারে আরও বিস্তারিত থাকবে। -(void)grab দিয়ে একটি grab নামের মেথড ডিক্লেয়ার করা হচ্ছে যার কোন প্যারামিটার নাই এবং এটা কোন কিছু রিটার্নও করে না। - সাইন দিয়ে এটা যে একটা instance method তা বোঝানো হয়। instance method এবং class method এর ব্যবহার সামনের চ্যাপ্টারে বিস্তারিত আসবে। বিঃ দ্রঃ অনেকেই এই ইন্টারফেস ফাইলেই কিছু প্রোটেক্টেড ডেরিয়েবল ডিফাইন করে থাকেন কিন্তু এটা ভালো প্র্যাকটিস নয়। এ ব্যাপারে আমাদের একদম প্রথম অর্থাৎ সিরিজ শুরুর আগের পোস্টেও লেখা ছিল। [এখানে](#)

objective-c তে একটি ক্লাসের ইন্টারফেস সাধারণত .h ফাইল-এ লেখা হয়। অন্যদিকে .m ফাইল এর টেমপ্লেট কোডেও ইন্টারফেস দেখা যায় যেটিকে মূলত বলা হয় ক্লাস এক্সটেনশন। .h ফাইলের এর ইন্টারফেসে সেই সব প্রপার্টি, মেথড ডিক্লেয়ার করা হয় যেগুলো হয়তবা অন্য ফাইল থেকেও ব্যবহার করা হতে পারে এবং যদি এমন কিছু প্রপার্টি, মেথড প্রয়োজন হয় যেগুলো শুধুই একটি .m ফাইলে ব্যবহৃত হবে অথবা প্রোটেক্টেড ডেরিয়েবল হিসেবে ব্যবহার করা হবে সেগুলোকে ক্লাস এক্সটেনশন এর মধ্যেই ডিক্লেয়ার করা ভালো অর্থাৎ ওই .m ফাইলের ইন্টারফেসের মধ্যে।

ইমপ্লেমেন্টেশনঃ নিচের কোড দেখে আমরা তার নিচে এটার বিভিন্ন লাইনের বর্ণনা করবো,

```
// food.m

#import "food.h"

@implementation food {
    // private instance variables here
    double makingCost;
}

- (void)grab {
    NSLog(@"Eating %@ :", self.item);
}

@end
```

@implementation ডিরেক্টিভ ব্যবহার করে ইমপ্লেমেন্টেশন তৈরি করা করা হয়। এখানে ইন্টারফেসের মত সুপার ক্লাসের নাম উল্লেখ করতে হয় না। এখানে একটি {} দিয়ে এর মধ্যে প্রয়োজনীয় প্রাইভেট ইন্সট্যান্স ডেরিয়েবল ডিক্লেয়ার করা হয়ে থাকে। এরপর grab ফাংশনের কার্যক্রম ডিফাইন করা হচ্ছে অর্থাৎ এই ফাংশন কোথাও থেকে কল হলে আসলে এর প্রেক্ষিতে কি ঘটবে। self হচ্ছে এখানে Java, C++ বা PHP এর this কি-ওয়ার্ড এর মত। এটা আসলে- এই মেথড যে ইন্সট্যান্স এর মাধ্যমে কল হয় তার রেফারেন্স। এই ক্লাসে যদি payBill নামের আরও একটি মেথড ডিফাইন করা থাকতো এবং সেটিকে এখানেই কল করার দরকার হলে [self payBill] লিখে কল করা যেত।

ইন্সট্যান্সিয়েট ও ব্যবহারঃ কোন ফাইলের যদি অন্য একটি ক্লাসের সাথে যোগাযোগের দরকার হয় তাহলে তাকে অবশ্যই ওই ক্লাসের হেডার/ইন্টারফেস ফাইল import করতে হবে। ওই ক্লাসের ইমপ্লেমেন্টেশন ফাইলকে সরাসরি এক্সেস করা উচিত হবে না। তা নাহলে এই যে, ইন্টারফেসের মাধ্যমে ইমপ্লেমেন্টেশন থেকে পাবলিক API আলাদা করে কাজ করার নিয়ম, এটাই ভেঙে যাবে। তা, এখন আমরা আমাদের main.m ফাইল থেকে এই ক্লাসের সাথে কাজ করব। এর জন্য নিচের মত করে main.m ফাইলকে আপডেট করতে হবে,

```
// main.m

#import
#import "food.h"

int main(int argc, const char * argv[])
{

    @autoreleasepool {

        food *pizza = [[food alloc] init];

        // accessing value using accessor methods aka getter/setter
        [pizza setItem:@"Italian Pizza"];
        NSLog(@"Ordered a %@", [pizza item]);

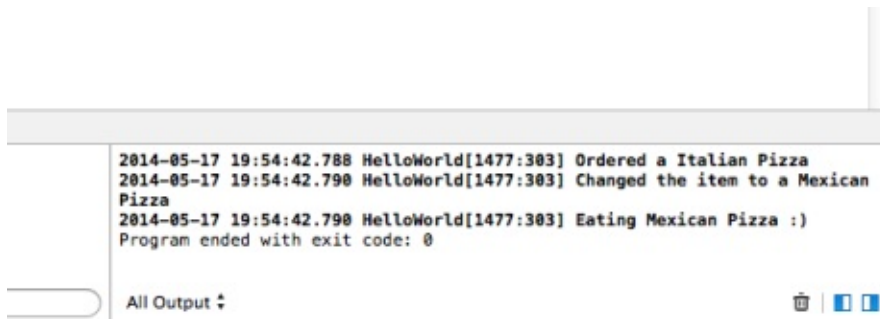
        // accessing value using dot syntax
        pizza.item = @"Mexican Pizza";
        NSLog(@"Changed the item to a %@", pizza.item);

        [pizza grab];

    }
    return 0;
}
```

৪ নম্বার লাইনে ইটারফেস import করার পর মেইন ফাংশনের মধ্যে থেকে food ক্লাসের অবজেক্ট ইন্সট্যান্সিয়েট করা হয়েছে alloc init প্যাটার্ন এর মাধ্যমে। অজেক্টিভ-সি তে একটি অবজেক্ট ইন্সট্যান্সিয়েট করা দুই ধাপে সম্পন্ন হয়। প্রথমে অবজেক্টটির জন্য কিছু মেমোরি অ্যালোকেট করে নেয়া হয় alloc নামের বিল্টইন মেথডের মাধ্যমে তারপর সেটিকে ইনিশিয়ালাইজ করা হয়। আর অবজেক্টটিকে একটি পয়েন্টার হিসেবে স্টোর করা হয় কারন ইতোমধ্যে অনেকবারই বলা হয়েছে যে অবজেক্টিভ-সি তে সব রকম অবজেক্টকেই পয়েন্টার হিসেবে স্টোর করতে হয়। আর তাই food pizza = ... না লিখে food *pizza = ... লেখা হয়েছে। এখানে food হচ্ছে যে ক্লাসের অবজেক্ট বানাতে চান তার নাম এবং pizza হচ্ছে অবজেক্টের নাম। এর পরেই এই অবজেক্টের প্রোপার্টি সেট করা হয়েছে item ডায়ালগবলের accessor method (গেটার/সেটার) ব্যবহার করে যেটা স্বয়ংক্রিয় ভাবেই হয়ে যায় Xcode 5 এ। এর আগে এই ডায়ালগবলকে @synthesize করে এর জন্য (গেটার/সেটার) জেনারেট করতে হত। গেটার হিসেবে শুধুমাত্র ডায়ালগবলটির নাম ব্যবহার করলেই হবে এবং সেটার হিসেবে ডায়ালগবলটির প্রথম অক্ষরকে বড় হাতের করে এবং সামনে set বসিয়ে দিয়ে এক্সেস করলেই হবে। যা করা হয়েছে উপরের ১৪, ১৫ নম্বার লাইনে। ১৪ নম্বারে ড্যালু সেট করা হয়েছে এবং ১৫ নম্বারে ড্যালু চেয়ে নেয়া হয়েছে। আবার, dot syntax ব্যবহার করেও এই অবজেক্টের প্রোপার্টি গুলোকে কিভাবে এক্সেস করা যেতে পারে তা দেখানো হয়েছে ১৮ এবং ১৯ নম্বার লাইনে। এরপরে ২১ নম্বার লাইনে [রিসিভার ম্যাসেজ] প্যাটার্ন মোতাবেক food অবজেক্টের grab মেথডকে কল করা হয়েছে। আর আলোচ্য অংশে যে, [] এর ব্যবহার সেটাকে অবজেক্টিভ-সি স্টাইল মনে করেই আগাতে পারেন। Command+R চেপে এই

অ্যাপকে রান করালে এর আউটপুট নিচের মতই আসবে এবং সেটা অ্যানালাইসিস করলেই বুঝতে পারবেন আপনি অবজেক্ট অরিয়েন্টেশনে ঢুকে পরেছেন।



ক্লাস মেথড ও ক্লাস ডেরিয়েবলঃ উপরের উদাহরণে ইন্সট্যান্স লেভেলের প্রোপার্টি ও মেথড নিয়ে কাজ করা হয়েছে। অর্থাৎ প্রথমে একটি ক্লাসের অবজেক্ট তৈরি করা হয়েছে এবং সেই অবজেক্টকে ধরেই ওই ক্লাসের প্রোপার্টি ও মেথড এর সাথে যোগাযোগ করা হয়েছে। অজেক্টিভ-সি তে ক্লাস লেভেল প্রোপার্টি ও মেথডও আছে। এর ডিক্লেয়ারেশনও ইন্সট্যান্স লেভেল মেথড এর মতই কিন্তু শুরুতে - সাইনের বদলে + সাইন দেয়া হয়। যেমন নিচে একটি ক্লাস লেভেল মেথড ডিক্লেয়ার করা হয়েছে, স্বভাবতই হেডার ফাইলে। উল্লেখ্য, এই নতুন মেথডটি কিন্তু একটি NSString টাইপের প্যারামিটার নেয় কিন্তু কিছু রিটার্ন করে না।

```
// food.h

#import

@interface food : NSObject

@property (copy) NSString *item;

-(void)grab;
+ (void)setDefaultItem:(NSString *)item;

@end
```

আর এই ধরনের মেথডের ইমপ্লিমেন্টেশনও একই রকম ভাবে করা হয়। নিচে দুইটি কাজ একসাথে করা হয়েছে প্রথমত একটি ক্লাস লেভেল মেথডের ইমপ্লিমেন্টেশন লেখা হয়েছে এবং দ্বিতীয়ত সেই মেথডের মাধ্যমে একটি স্ট্যাটিক ড্যারিয়েবলের ভ্যালু সেট করা হয়েছে যেহেতু অজেক্টিভ-সি তে ক্লাস লেভেল ড্যারিয়েবল বলে কিছু নাই। অর্থাৎ defaultItem কে আমরা অন্য কোথাও থেকে ক্লাস লেভেল ড্যারিয়েবল হিসেবে এক্সেস করতে পারবো না যেমন ভাবে setDefaultItem মেথড কে ক্লাস লেভেল মেথড হিসেবে এক্সেস করতে পারবো।

```
// food.m

#import "food.h"

static NSString *defaultItem;

@implementation food {
    // private instance variables here
    double makingCost;
}

- (void)grab {
    NSLog(@"Eating %@ with an obvious %@", self.item, defaultItem);
}

+ (void)setDefaultItem:(NSString *)item {
    defaultItem = [item copy];
}

@end
```

এখন main.m ফাইল থেকে আমরা নিচের মত করে এই ক্লাস লেভেল মেথডটিকে এক্সেস করতে পারি (১১ নম্বর লাইনে সরাসরি ক্লাসের নাম দিয়েই ওটার একটি মেথড setDefaultItem কে ধরা যাচ্ছে) যেটা কিনা এর কাছে পাঠানো একটি স্ট্রিংকে ওই ক্লাসের একটি স্ট্যাটিক ভ্যারিয়েবলের ভ্যালু হিসেবে সেট করে। তারপর আমরা আগের উদাহরণে যা করেছিলাম ঠিক সেই কাজ আবার করেছি।


```
// main.m

#import
#import "food.h"

int main(int argc, const char * argv[])
{
    @autoreleasepool {

        [food setDefaultItem:@"Lemonade"];

        food *pizza = [[food alloc] init];

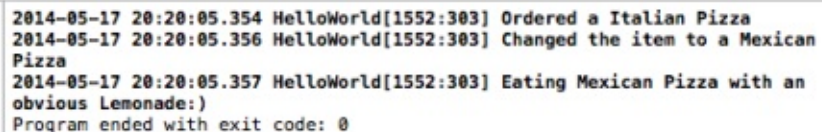
        // accessing value using accessor methods aka getter/setter
        [pizza setItem:@"Italian Pizza"];
        NSLog(@"Ordered a %@", [pizza item]);

        // accessing value using dot syntax
        pizza.item = @"Mexican Pizza";
        NSLog(@"Changed the item to a %@", pizza.item);

        [pizza grab];

    }
    return 0;
}
```

কিন্তু এবার grab মেথড জানাচ্ছে যে, আপনি একটা পিৎজা খাচ্ছেন কিন্তু সাথে আপনার একটা কমন পছন্দের



```
2014-05-17 20:20:05.354 HelloWorld[1552:303] Ordered a Italian Pizza
2014-05-17 20:20:05.356 HelloWorld[1552:303] Changed the item to a Mexican
Pizza
2014-05-17 20:20:05.357 HelloWorld[1552:303] Eating Mexican Pizza with an
obvious Lemonade:)
Program ended with exit code: 0
```

আইটেম আছে লেমনেড।

কম্পট্রাক্টরঃ অবজেক্টিভ-সি তে সরাসরি কোন কম্পট্রাক্টর মেথড নাই। তবে অন্যান্য ল্যাঙ্গুয়েজের বিস্টইন কম্পট্রাক্টর মেথড যা করে (ক্লাস থেকে অবজেক্ট ইন্সট্যান্সিয়েট করার সময়ই কিছু সাধারণ কাজ করে ফেলা) ঠিক তাই করা যাবে নিচের পদ্ধতি অনুযায়ী। অবজেক্টিভ-সি তে একটি অবজেক্ট অ্যালোকেটেড হবার পর পরই init মেথড কল করার মাধ্যমে সেটি ইনিশিয়ালাইজ হয়। যেমন উপরের উদাহরণে food *pizza = [[food alloc] init] লাইনে। আবার ক্লাস লেভেল ইনিশিয়ালাইজেশনও আছে সেটা পরে দেখা যাবে। আগে এই পদ্ধতিতে কম্পট্রাকশন এর কাজ করি। init হচ্ছে একটি ডিফল্ট ইনিশিয়ালাইজেশন মেথড। কিন্তু আপনি নিজেও আপনার মত করে একটা ইনিশিয়ালাইজেশন মেথড তৈরি করতে পারেন। এটা খুব একটা কঠিন কাজ না, শুধু নরমাল টাইপের একটা মেথডের নামের আগে init শব্দটা ব্যবহার করতে হবে। নিচে আমরা সেরকম একটি ডিক্লেয়ার করেছি food.h ফাইলে যার নাম initWithItem,

```
// food.h

#import

@interface food : NSObject

@property (copy) NSString *item;

-(void)grab;
-(id)initWithItem:(NSString *)item;

@end
```

এখন এই কাস্টম ইনিশিয়ালাইজার এর ইমপ্লেমেন্টেশন করতে হবে নিচের মত করে,

```
// food.m

#import "food.h"
#import "food.h"

@implementation food {

}

- (void)grab {
    NSLog(@"Eating %@ :", self.item);
}

- (id)initWithItem:(NSString *)item {
    self = [super init];
    if (self) {
        _item = [item copy];
    }
    return self;
}

@end
```

এখানে super, প্যারেন্ট ক্লাসকে নির্দেশ করে এবং self হচ্ছে সেই ইন্সট্যান্সকে/অবজেক্টকে নির্দেশ করে যেটা এই মেথডটিকে কল করে। ইনিশিয়ালাইজার মেথডকে সবসময় ওই অবজেক্টের কাছে নিজের রেফারেন্সটাই রিটার্ন করতে হয়। আর তাই, এই মেথডের মধ্যে অন্যান্য কাজ করার আগে চেক করে নিতে হয় self আছে কিনা। তারপর আমরা ডাইরেক্ট `_item = ...` ব্যবহার করে এই ইন্সট্যান্স ভ্যারিয়েবলের ভ্যালু সেট করে দিলাম এই মেথডের সাথে আসা প্যারামিটার এর কন্টেন্টকে কপি করে। অর্থাৎ যখনই initWithItem ব্যবহার করে এই ক্লাস এর অবজেক্ট ইন্সট্যান্সিয়েট করা হচ্ছে ঠিক তখনই এই ক্লাসের একটি প্রোপারটির ভ্যালু আমরা আসাইন করে দিচ্ছি। ঠিক এটাই সহজ ভাবে যেকোনো কন্সট্রাক্টর মেথডের কাজ। এখন এটা টেস্ট করার জন্য main.m ফাইলটিকে নিচের মত করে পরিবর্তন করে নিতে পারেন,


```
// main.m

#import
#import "food.h"

int main(int argc, const char * argv[])
{
    @autoreleasepool {
        food *choice = [[food alloc] initWithItem:@"Pan Pizza"];

        [choice grab];
    }
    return 0;
}
```

উপরের কোড ব্লক লক্ষ্য করলে দেখবেন যে, food ক্লাসের অবজেক্ট choice কে ইনিশিয়ালাইজ করা হয়েছে initWithItem নামক কাস্টম ইনিশিয়ালাইজার মেথড দিয়ে। আর এই initWithItem মেথড food ক্লাসের একটি প্রোপার্টি যার নাম item সেটার ভ্যালু সেট করে দেয়। আর তাই, অবজেক্ট ইনিশিয়ালিয়াজেশনের পর পরই যদি আমরা grab মেথড কল করি যেটা কিনা কনসোলে ওই আইটেমটা খাওয়ার কথাটাই প্রিন্ট করার কথা - তাহলে নিচের মত



```
2014-05-18 02:27:43.904 HelloWorld[384:303] Eating Pan Pizza :)
Program ended with exit code: 0
```

ঠিক ঠাক আউটপুট আসে।

পরের চ্যাপ্টারঃ প্রোপার্টি (@property) নিয়ে বিস্তারিত আলোচনা এবং বিভিন্ন অ্যাট্রিবিউট সাথে এর সম্পর্ক নিয়ে কিছু উদাহরণ এবং আলোচনা থাকবে।

Originally Posted [Here](#)

প্রোপার্টি ও এর অ্যাট্রিবিউটের ব্যবহার

ভূমিকাঃ গত অধ্যায়ে অবজেকটিভ সি তে অবজেক্ট অরিয়েন্টেড প্রোগ্রামিং এর প্রয়োগ নিয়া আলোচনা হয়েছে। প্রোপার্টি, মেথড, ক্লাস, ইন্টারফেস, ইমপ্লিমেন্টেশন, অবজেক্ট ইন্সট্যান্সিয়েট ও সেগুলোর ব্যবহার সম্পর্কে ধারণা পেয়েছি আমরা। এই অধ্যায়ে প্রোপার্টি ও এর অ্যাট্রিবিউট গুলোর ব্যবহার নিয়ে বিস্তারিত আলোচনা হবে সাথে মেমোরি ম্যানেজমেন্ট নিয়েও প্রাসঙ্গিক আলোচনা চলে আসবে।

সাধারণত, একটি অবজেক্টের প্রোপার্টিগুলো অন্য আরেকটি অবজেক্টকে তাদের অবস্থা পরিবর্তন এবং যাচাই করতে দেয়। কিন্তু ভালভাবে ডিজাইন করা কোন অবজেক্ট অরিয়েন্টেড প্রোগ্রামে অবজেক্টের ইন্টারনাল অবস্থা জানা বা পরিবর্তন করা সহজ নয়। এর জন্য প্রয়োজন Accessor Methods তথা getters এবং setters।

Objective C তে @property এর কাজ হল অটোমেটিক্যালি এই Accessor method গুলো জেনারেট করে সহজে প্রোপার্টি তৈরী এবং কনফিগার করার ব্যবস্থা করা। যেসব প্রোপার্টি গুলো Public হবে শুধুমাত্র সেগুলোর জন্যই @property ডিরেকটিভ ব্যবহার করা হয়। ফলে সহজেই সিমাল্টিক লেভেলে প্রোপার্টির behavior ঠিক করে দেওয়া যায় এবং এটা একই সাথে ওই প্রোপার্টির ইম্পলিমেন্টেশন ডিটেইলসও ম্যানেজ করে।

@property ডিরেকটিভঃ প্রথমেই দেখা যাক @property ডিরেকটিভ ব্যবহার করলে কি ঘটনা ঘটে। গত অধ্যায়ের সাথে মিল রেখে একটি Food ক্লাসের জন্য একটি সহজ ইন্টারফেস এবং ইম্পলিমেন্টেশন লেখা যাক।

```
// Food.h

#import

@interface Food : NSObject

@property NSString *item;

@end
```

```
// Food.m

#import "Food.h"

@implementation Food

@end
```

Food ইন্টারফেসের একটি প্রোপার্টি item যার ডাটাইটাইপ NSString। যেহেতু অবজেকটিভ সি তে সকল ডায়রিয়েবলকে পয়েন্টার হিসেবে ব্যবহার করা হয় তাই item এর আগে পয়েন্টার চিহ্ন ব্যবহার করা হয়েছে। লক্ষণীয় ব্যপার হল এখানে item ডিক্লেয়ার করার আগে @property ডিরেকটিভ লেখা আছে।

কোন ভ্যারিয়েবলকে @property ডিরেকটিভ হিসেবে ডিক্লেয়ার করা হলে, তার জন্য অটোমেটিক্যালি Accessor methods জেনারেট হয়ে যায় এবং নিজ ক্লাসের ইন্টারফেস ও ইমপ্লিমেন্টেশনে থাকা অন্যান্য মেথড গুলোর মতই ওই অদৃশ্য গেটার এবং সেটার মেথড গুলোকে কল করা যায়। item প্রোপারটির জন্য আসলে নিচের মত গেটার এবং সেটার মেথড তৈরি হয়ে গেছে যেগুলো আমরা একটু পরেই মূল প্রোগ্রামে ব্যবহার করতে পারবো। বিঃ দ্রঃ কাস্টম সেটার ও গেটার তৈরির করার জন্য Food.m -এ এই গেটার এবং সেটারকে ওভাররাইডও করা যায়। তবে কাস্টম সেটার ও গেটার তৈরির জন্য @synthesize ডিরেকটিভ আবশ্যিক।

```
- (NSString *)item {
    return item;
}
- (void)setItem:(NSString *)newValue {
    item = newValue;
}
```

এবার আমাদের main.m ফাইলকে নিচের মত করে আপডেট করুন।

```
// main.m

#import
#import "Food.h"

int main(int argc, const char * argv[])
{
    @autoreleasepool {

        Food *food = [[Food alloc] init];
        food.item = @"A bengali dish!"; // Under hood: [food setItem: @"A bengali dish!"]

        NSLog(@"Eating a %@", food.item); // Under hood: [food item]

    }
    return 0;
}
```

ডট নোটেশন দিয়ে প্রোপারটি এক্সেস করার সময় মূলত ব্যাকএন্ডে accessor মেথড গুলোতেই ট্রান্সলেটেড হয়। ফলে food.item এ কোন স্ট্রিং এসাইন করলে setItem: মেথড কল হয় এবং food.item দিয়ে ডাটা চাইলে item: মেথডটাই কল হয়।

এক্সেসর মেথডগুলোর বিহেভিয়ার পরিবর্তন করার জন্য @property ডিরেকটিভ এর পর প্যারেনথেসিস () এর ভিতরে বিহেভিয়ার এর ধরন তথা এটিবিউট সেট করা যায়। এই চাপ্টারের বাকি অংশে বিভিন্ন রকম অ্যাট্রিবিউট নিয়ে আলোচনা হবে।

getter= এবং **setter=** অ্যাট্রিবিউটস্‌; @property ডিরেকটিভ এর ডিফল্ট নেমিং কনভেনশন পছন্দ না হলে, আমরা getter= এবং setter= ব্যবহার করে getter এবং setter মেথড গুলোর নাম পরিবর্তন করতে পারি নিচের মত করে। উল্লেখ্য, গেটার/সেটারকে কাস্টম নাম দেয়ার একটা সাধারণ সিচুয়েশন হচ্ছে যখন কোন বুলিয়ান টাইপ প্রোপার্টির ব্যবহার চলে আসে এবং সেগুলোকে আরও হিউম্যান রিডেবল করার জন্য।

```
@property (getter=isOpen) BOOL open;
```

এ অবস্থায় জেনারেট হওয়া এক্সেসর মেথড গুলোকে isOpen এবং setOpen দিয়ে কল করা হবে/যাবে। আসুন আমাদের Food.h, Food.m এবং main.m কে নিচের মত করে সাজাই,

```
// Food.h

#import

@interface Food : NSObject

@property (getter=isOpen) BOOL open;

@end
```

```
// Food.m

#import "Food.h"

@implementation Food

@end
```

```
// main.m

#import
#import "Food.h"

int main(int argc, const char * argv[])
{
    @autoreleasepool {

        Food *place = [[Food alloc] init];
        place.open = NO; // Under hood: [food setOpen]

        // NSLog(@"Is the restaurant open?: %hhd", [place open]); // Error: No method exi

        NSLog(@"Is the restaurant open?: %hhd", [place isOpen]); // Works!

        NSLog(@"Is the restaurant open?: %hhd", place.open); // Works! Under hood: [plac

    }
    return 0;
}
```

খেয়াল করুন এখন কিন্তু আর গেটারের ব্যাকএন্ড স্টাইল `[place open]` কাজ করবে না বরং `[place isOpen]` কাজ করবে যেখানে `isOpen` কে গেটার হিসেবে বলে দিয়েছি আমরা। আবার, এখনো কিন্তু পাবলিক প্রোপার্টিগুলোকে শুধু প্রোপার্টির নাম দিয়েও এক্সেস করা যাচ্ছে। বিঃ দ্রঃ শুধুমাত্র এই অ্যাট্রিবিউট দুটি আর্গুমেন্ট এক্সপেক্ট করে। অন্য সকল অ্যাট্রিবিউটগুলো বুলিয়ান ফ্ল্যাগ।

রিডঅনলি (**readonly**) অ্যাট্রিবিউটঃ ক্লাসের কোন প্রোপার্টিকে রিড অনলি (শুধু ড্যালাু রিড করা যাবে, নতুন করে সেট করা যাবে না) করার সহজ উপায় হল এই প্রোপার্টির জন্য `readonly` অ্যাট্রিবিউট সেট করা। কোন প্রোপার্টিকে `readonly` করলে, প্রোপার্টির `setter` মেথড থাকে না এবং ডট নোটেশন ব্যবহার করে ড্যালাু অ্যাসাইন করা যায় না। নিচের উদাহরণটি দেখলেই পরিষ্কার হয়ে যাবে।

```
// Food.h

#import

@interface Food : NSObject

@property (getter=isOpen, readonly) BOOL open;

@end
```

```
// Food.m

#import "Food.h"

@implementation Food

@end
```

```
// main.m

#import
#import "Food.h"

int main(int argc, const char * argv[])
{

    @autoreleasepool {

        Food *place = [[Food alloc] init];
        place.open = YES; // Error: can't assign to readonly property

    }
    return 0;
}
```

এ অবস্থায় Xcode -ই আপনাকে আটকিয়ে দিবে নিচের মত,



তবে হ্যাঁ, আমরা ইন্সট্যান্স মেথড ব্যবহার করে ইন্টারনালি এই open নামক রিডঅনলি প্রোপারটির ড্যালা পরিবর্তন করতে পারবো। এই সিকুয়েন্স টাই করার জন্য নিচে যথাক্রমে ইন্টারফেস, ইমপ্লিমেন্টেশন ও মেইন ফাইলটি দেওয়া হল:


```
// Food.h

#import

@interface Food : NSObject

@property (getter=isOpen, readonly) BOOL open;
-(void)openTheRestaurant;

@end
```

```
// Food.m

#import "Food.h"

@implementation Food

-(void)openTheRestaurant
{
    // N.B. We can use _open directly, cause @property also creates an
    // instance variable for us!
    _open = YES;
}

@end
```

```
// main.m

#import
#import "Food.h"

int main(int argc, const char * argv[]) {
    @autoreleasepool {

        Food *place = [[Food alloc] init];
        [place openTheRestaurant]; // Setting the new value by this imethod

        NSLog(@"Is the restaurant open?: %hhd", place.open);
    }
    return 0;
}
```

এটি রান করলে কনসোলে Is the restaurant open?:1 দেখতে পাবেন অর্থাৎ ওই রিডঅনলি বুলিয়ান প্রোপার্টিটার ভ্যালু চেঞ্জ করা গেছে যেটা বাই ডিফল্ট 0 থাকে।

ননঅ্যাটমিক (**nonatomic**) অ্যাট্রিবিউটঃ মাল্টিথ্রেডেড এনভায়রনমেন্টে (যখন কোন অ্যাপ্লিকেশনে একাধিক থ্রেড থাকে) getter এবং setter মেথড গুলো একাধিকবার কল হতে পারে। অর্থাৎ অন্য কোন অপারেশন বর্তমানে এক্সিকিউট হচ্ছে এমন কোন getter/setter মেথডের এক্সিকিউশনে ইন্টারাপ্ট করে রেজাল্টে প্রভাব ফেলতে পারে। Atomic প্রোপার্টি অবজেক্টকে লক করে দেয় যাতে করে getter/setter মেথড গুলো কমপ্লিট ড্যালা নিয়ে কাজ করতে পারে।

@property ডিরেকটিভ দিয়ে ডিক্লেয়ার করা প্রোপার্টি বাই ডিফল্ট Atomic হয়। যা আমাদের মাথার উপর থেকে ম্যানুয়ালী Atomicity নিশ্চিত করার বোঝা কমিয়ে দিয়েছে। কিন্তু যদি আপনার অ্যাপ্লিকেশনটি একটিমাত্র থ্রেড নিয়ে হয় অথবা আপনি নিজের মত করে Thread-Safety ইমপ্লিমেন্ট করে থাকেন, তাহলে প্রোপার্টিটিকে nonatomic হিসেবে ডিক্লেয়ার করলেই চলে।

```
@property (nonatomic) NSString *item;
```

আরও একটি উল্লেখযোগ্য ব্যাপার হল, Atomic প্রোপার্টিগুলোর getter এবং setter দুটি মেথডই অটো-জেনারেটেড হবে অথবা দুটি মেথডই ইউজার ডিফাইন্ড হবে। কিন্তু nonatomic প্রোপার্টি গুলোর জন্য কোন একটি ইউজার ডিফাইন্ড এবং অন্যটি অটো জেনারেটেড এরকম মিক্স হতে পারে।

মেমরী ম্যানেজমেন্ট (**Memory Management**)ঃ অবজেক্ট অরিয়েন্টেড ল্যাঙ্গুয়েজে, অবজেক্ট গুলো কম্পিউটারের মেমরী লোকেশনে স্টোর করা হয়। এখনো পর্যন্ত মেমরীই হল দুর্লভ রিসোর্স বিশেষ করে মোবাইল ডিভাইস গুলোর জন্য। মেমরী ম্যানেজমেন্টের মূল লক্ষ্যই হল কম্পিউটার বা মোবাইলের মেমরীর সঠিক ব্যবহার করা। অ্যাপ্লিকেশনটি অপয়োজনীয় কোন মেমরী যাতে ব্যবহার করতে না পারে সেটা নিশ্চিত করাই মেমরী ম্যানেজমেন্টের কাজ।

বেশীরভাগ ল্যাঙ্গুয়েজেই garbage collection দিয়েই মেমরী ম্যানেজমেন্ট করা হয়। কিন্তু অবজেকটিভ সি (Objective C) অবজেক্ট ওনারশিপ (Object Ownership) দিয়ে মেমরী ম্যানেজ করে যা garbage collection এর চেয়ে অনেকবেশী ইফিশিয়েন্ট। যখন কোন প্রোগ্রাম কোন অবজেক্টের সাথে ইন্টারঅ্যাক্ট শুরু করে তখনই প্রোগ্রামটিকে এই অবজেক্টের ওনার (owner) বলা হয়। আবার যখন প্রোগ্রামটির কাজ শেষ হয়ে যায় অথবা প্রোগ্রামটির আর অবজেক্টের দরকার থাকে না তখন অবজেক্টটির owner হিসেবে প্রোগ্রামটি থাকেনা। একটি অবজেক্টের একাধিক owner থাকতে পারে। অর্থাৎ যখন কোন অবজেক্টের কমপক্ষে একটি ওনার থাকবে তখন বুঝতে হবে অবজেক্টটি ব্যবহার করা হচ্ছে। কোন অবজেক্টের যদি একটিও Owner না থাকে তখন বুঝতে হবে যে এই অবজেক্টের আর কোন কাজ নেই। তাই অপারেটিং সিস্টেম অবজেক্টটিকে ডেস্ট্রয় করে দিয়ে মেমরী ফ্রি করে নেয়।

AUTOMATIC REFERENCE COUNTING এর মাধ্যমে অপারেটিং সিস্টেম নিজে নিজেই মেমরী ম্যানেজমেন্টের কাজ গুলো করে নেয়। এসব নিয়ে আমাদের মাথা গরম করার কোন দরকার নেই। কিন্তু আমাদেরকে অবশ্যই strong, weak এবং copy অ্যাট্রিবিউটগুলো সম্পর্কে জানতে হবে কারণ এই অ্যাট্রিবিউটগুলোই কম্পাইলারকে অবজেক্টের রিলাশনশিপ কেমন হবে তা জানায়।

স্ট্রং (**strong**) এবং উইক (**weak**) অ্যাট্রিবিউটঃ প্রথমেই দেখি কিভাবে একটি স্ট্রং অ্যাট্রিবিউটের প্রোপার্টি ডিক্লেয়ার করা যায়,

```
@property (nonatomic, strong) NSString *name;
```

আগেও একবার বলা হয়েছে যে, ARC বা Automatic Reference Counting হচ্ছে একটি স্বয়ংক্রিয় মেমোরি ম্যানেজমেন্ট ব্যবস্থা যার মাধ্যমে সিস্টেম স্বয়ংক্রিয় ভাবে যেকোনো অপ্রয়োজনীয় অবজেক্টের মেমোরি ফ্রি করে। আর এই ফ্রি করার ব্যাপারটা সে ঠিক করে রেফারেন্সের নম্বর (Count) এর উপর। যেটার রেফারেন্স যখন 0 হয়ে যায় তখনই তার মেমোরি ফ্রি/রিলিজ করে দেয়।

উপরে strong অ্যাট্রিবিউট হচ্ছে কোন অবজেক্টের এমন একটি স্ট্রং রেফারেন্স যা তাকে ডিঅ্যালোকেটেড হতে দেয় না। অর্থাৎ তার রেফারেন্স কাউন্ট হয় 0 এর চেয়ে বেশি বা 1, 2 ... এরকম। এখানে name কে strong প্রোপার্টি হিসেবে ডিক্লেয়ার করা দুটি অর্থ বহন করে: ১। যখন আপনি name এর মধ্যে একটি অবজেক্ট অ্যাসাইন করবেন, তখন সেই অবজেক্টটির রেফারেন্স কাউন্ট বেড়ে যায়। ২। যখন অ্যাসাইন করা অবজেক্টটি যাকে name পয়েন্ট করে সেটাকে আপনি বা সিস্টেম কোনভাবে ডিঅ্যালোকেট করবেন তখনও name এর ড্যান্সু আগের মতই থাকবে অর্থাৎ name আসলে এর মধ্যে অ্যাসাইন করা অবজেক্টটির মালিকানা পেয়ে গিয়েছিল; কারণ name একটি strong Relationship.

এবার আসুন একটি উদাহরণ দেখি। প্রথমেই আমাদের বর্তমান প্রজেক্টে আরও একটি নতুন ক্লাস (NSObject এর সাব) যুক্ত করে নিন যার নাম Person. ফলে নতুন দুটি ফাইল পেয়ে যাবো আমরা যেগুলোর কোড আপডেট করে নিন নিচের মত,

```
// Person.h

#import

@interface Person : NSObject

@property (nonatomic) NSString *name;

@end
```

অর্থাৎ এই নতুন ক্লাসের একটি মাত্র সাধারণ প্রোপার্টি যার নাম name.

```
// Person.m

#import "Person.h"

@implementation Person

- (NSString *)description {
    return self.name;
}

@end
```

Person ক্লাসের ইমপ্লিমেন্টেশনে আমরা description নামের একটি ডিফল্ট মেথডকে ওভাররাইড করেছি যার কাজ হল এর name প্রোপার্টিটিকে রিটার্ন করা। অর্থাৎ কোথাও এই ক্লাসের ডাইরেক্ট অবজেক্টকেই ব্যবহার করলে তা পক্ষান্তরে এটার name প্রোপার্টিটিকেই সেখানে হাজির করবে। এটুকু আমরা করছি আমাদের উদাহরণ এর স্বার্থে, description মেথডের ব্যবহার শেখাতে নয়।

এখন নিচের মত করে Food.h ফাইল আপডেট করুন,

```
// Food.h

#import
#import "Person.h"

@interface Food : NSObject

@property (nonatomic) NSString *item;
@property (nonatomic, strong) Person *cook;

@end
```

এখানে প্রথমে আমরা একটি সাধারণ NSString প্রোপার্টি অ্যাড করেছি এবং তার নিচে একটি Person প্রোপার্টি অ্যাড করেছি এবং সেটার অ্যাট্রিবিউট হিসেবে strong সেট করেছি। অর্থাৎ পরবর্তীতে এই cook এর মধ্যে যখন আমরা কোন অবজেক্ট অ্যাসাইন করব তখন সেই অবজেক্টটির রেফারেন্স কাউন্ট বেড়ে যাবে।

Food এর ইমপ্লিমেন্টেশন একদমই ফাকা আপাতত এই উদাহরণ এর সাপেক্ষে,

```
// Food.m

#import "Food.h"

@implementation Food

@end
```

এবার main.m কে নিচের মত আপডেট করুন,

```
// main.m

#import
#import "Food.h"

int main(int argc, const char * argv[])
{

    @autoreleasepool {

        Person *tomi = [[Person alloc] init];
        tomi.name = @"Tomi Mia";

        Food *newDish = [[Food alloc] init];
        newDish.item = @"Halim";
        newDish.cook = tomi;

        NSLog(@"%@ is cooking the %@", newDish.cook, newDish.item);

    }
    return 0;
}
```

খেয়াল করুন, প্রথমে আমরা Person ক্লাস ইন্সট্যান্সিয়েট করে tomi অবজেক্টের মাধ্যমে এর (Person এর) name প্রোপার্টিটি সেট করে দিয়েছি। তারপর আমরা Food ক্লাস ইন্সট্যান্সিয়েট করে newDish অবজেক্টের মাধ্যমে এর (Food এর) item প্রোপার্টিটি সেট করে দিয়েছি। অতঃপর, newDish অবজেক্টের মাধ্যমে এর cook নামের স্ট্রিং প্রোপার্টিটির ভেতর আমাদের একটু আগে তৈরি করা tomi অবজেক্টটিকে অ্যাসাইন করেছি। এই প্রোগ্রামকে রান করলে কনসোলে আউটপুট আসবে "Tomi Mia is cooking the Halim"। যেহেতু, cook একটি strong relationship তাই newDish অবজেক্টটি tomi এর ownership নিয়ে নেয় এবং এটা ততক্ষণ পর্যন্ত ভ্যালিড থাকে যতক্ষণ পর্যন্ত tomi কে newDish এর প্রয়োজন হয়।

এখন সবকিছু ঠিক রেখেই শুধুমাত্র main.m কে নিচের মত করে আপডেট করুন,

```
// main.m

#import
#import "Food.h"

int main(int argc, const char * argv[])
{

    @autoreleasepool {

        Person *tomi = [[Person alloc] init];
        tomi.name = @"Tomi Mia";

        Food *newDish = [[Food alloc] init];
        newDish.item = @"Halim";
        newDish.cook = tomi;

        tomi = nil; // Trying to deallocating the "tomi" object

        NSLog(@"%@ is cooking the %@", newDish.cook, newDish.item);

    }
    return 0;
}
```

এখানে আমরা একটা ধাপে tomi অবজেক্টকে ম্যানুয়ালি nil করেছি কিন্তু প্রোগ্রামটি রান করে দেখুন আউটপুট আগের মতই আসবে, "Tomi Mia is cooking the Halim"।

কিন্তু,

এবার নিচের মত করে Food.h ফাইলকে আপডেট করুন,

```
// Food.h

#import
#import "Person.h"

@interface Food : NSObject

@property (nonatomic) NSString *item;
@property (nonatomic, weak) Person *cook;

@end
```

অর্থাৎ cook কে weak করে ফেলুন এবং main.m রাখুন একটু আগে যেমন ছিল তেমনি,

```
// main.m

#import
#import "Food.h"

int main(int argc, const char * argv[])
{

    @autoreleasepool {

        Person *tomi = [[Person alloc] init];
        tomi.name = @"Tomi Mia";

        Food *newDish = [[Food alloc] init];
        newDish.item = @"Halim";
        newDish.cook = tomi;

        tomi = nil; // Trying to deallocating the "tomi" object

        NSLog(@"%@ is cooking the %@", newDish.cook, newDish.item);

    }
    return 0;
}
```

এবার প্রোগ্রামটিকে রান করলে আউটপুট আসবে "(null) is cooking the Halim"। অর্থাৎ tomi অবজেক্ট ডিঅ্যালোকেট হবার পর newDish এর weak প্রোপার্টি cook, tomi কে হারিয়ে ফেলছে। কিন্তু ১৮ নম্বার লাইনটি মুছে ফেলে দেখুন, কনসোলে আগের মত আউটপুট আসবে।

আশা করি স্ট্রং এবং উইক প্রোপার্টি এর ব্যপারগুলো একটু হলেও বোঝানো গেছে। আমাদের সম্ভাব্য বাংলা কাণ্ডজে বইয়ে আরও বিস্তারিত আলোচনা থাকবে এই ব্যাপারে।

কপি (**copy**) অ্যাট্রিবিউটঃ এটি মোটামুটি strong এর মতই কিন্তু বিশাল একটা পার্থক্য আছে। স্ট্রং এর মত সরাসরি অবজেক্টের ownership না নেয়ার বদলে এটি প্রথমে, ওই প্রোপার্টির মধ্যে যা অ্যাসাইন করা হয় সেটার একটা কপি তৈরি করে এবং সেই কপিটির ownership নিয়ে রাখে। যে ধরনের প্রোপার্টি গুলো ভ্যালু রিপ্রেজেন্ট করে সেগুলোই সাধারণত copy হিসেবে ডিক্লেয়ার করা হয় যেমন NSString টাইপের প্রোপার্টি গুলো।

```
@property (nonatomic, copy) NSString *item;
```

আরও যেসব অ্যাট্রিবিউট রয়েছেঃ

- retain
- unsafe_unretained
- assign

পরের চ্যাপ্টারঃ পরের চ্যাপ্টারে থাকছে মেথড নিয়ে বিস্তারিত আলোচনা। এদের নেমিং কনভেনশন, ডাইনামিক মেথড কলিং ইত্যাদি।

Originally Posted [Here](#)

মেথড ও এর বিস্তারিত

ভূমিকাঃ মেথড যেকোনো কিছু কার্যপ্রণালি বর্ণনা করে এবং সেটা কখন ঘটাতে হবে তা নির্ভর করে একটি অবজেক্ট এর উপর যার মাধ্যমে ওই মেথডকে প্রয়োজন অনুযায়ী কল করা হয়। এই চ্যাপ্টারে আমরা অবজেক্টিভ সি তে মেথডের নেমিং কনভেনশন, ব্যবহার এবং এক্সেস মডিফায়ার নিয়ে আলোচনা করব। আরও থাকবে, কিভাবে সিলেক্টর ব্যবহার করে মেথডের রেফারেন্স করা যায় এবং ডাইনামিক্যালি মেথড কল করা যায়।

নেমিং কনভেনশনঃ সাধারণত অবজেক্টিভ সি তে মেথডের নামকরণ করার সময় বেশি বেশি শব্দ ব্যবহার করা হয় বলে আপাত দৃষ্টিতে মনে হয় কিন্তু সত্যি বলতে এই পদ্ধতিতে মেথডটি অনেক বেশি বর্ণনামূলক ও বোধগম্য হয়। কোন রকম শব্দের সংক্ষেপণ না করা এবং বিশদভাবে মেথডটির প্যারামিটার ও রিটার্ন ভ্যালু ঠিক করে দেয়াই হচ্ছে অবজেক্টিভ সি তে মেথডের নামকরণের অঘোষিত নিয়ম।

নিচে কিছু উদাহরণ দেয়া হল,

```
// Calculated values
- (double)maximumPreparationTime;
- (double)maximumPreparationTimeUsingTool:(NSString *)tool;

// Action methods
- (void)startCooking;
- (void)cookFixedSize:(double)theSize;
- (void)makeWithType:(NSString *) type withSize:(int) size;

// Custom Constructor methods
- (id)initWithItem:(NSString *)item;
- (id)initWithItem:(NSString *)item size:(double)theSize;

// Factory methods
+ (Food *)food;
+ (Food *)foodWithItem:(NSString *)item;
```

মেথডের নাম ও কাজকে সহজ বোধগম্য করার সবচেয়ে সাধারণ উপায় হচ্ছে এর নামের শব্দ গুলোকে সংক্ষেপ না করা। যেমন উপরে একটি মেথডের নাম লেখা হয়েছে maximumPreparationTime যেটাকে হয়তবা এই নিয়মের বাইরে গিয়ে maxPrepTime ও লেখা যেত।

C++ বা Java তে যেখানে মেথড এবং এর প্যারামিটার গুলোকে আলাদা ভাবেই ট্রিট করা হয়, সেখানে অবজেক্টিভ সি তে একটি মেথডের নামের সাথে এর প্যারামিটার গুলোর নামও যুক্ত থাকে। যেমন উপরের initWithItem:size: মেথডটি দুইটি প্যারামিটার গ্রহণ করে আর তাই এই মেথডের নাম মূলত initWithItem:size: এটাই। শুধুমাত্র initWithItem: নয়। এখানে initWithItem:size: নামটি প্রথম আর্গুমেন্ট হিসেবে item এবং দ্বিতীয় আর্গুমেন্ট হিসেবে size প্রকাশ করছে।

আরও একটি উল্লেখযোগ্য বিষয় হচ্ছে মেথড গুলোর রিটার্ন করা ভ্যালু আসলেই কি জিনিস সেটাও মেথড ডিক্লেয়ার করার সময়ই পরিষ্কার ভাবে ঠিক করে দেয়া হয়। যেমন উপরের ফ্যাক্টরি মেথড গুলো দেখে বোঝা যাচ্ছে যে, এগুলো Food ক্লাসের ইন্সট্যান্স রিটার্ন করবে।

মেথড কল করাঃ একটি স্বয়ার ব্রাকেটের মধ্যে প্রথমে অবজেক্টের নাম এবং একটা স্পেস দিয়ে মেথডের নাম লিখে যেকোনো অবজেক্টের মেথডকে কল করা হয়। প্যারামিটার ওয়ালা মেথডের ক্ষেত্রে একটি কোলন দিয়ে মেথডের নাম এবং আর্গুমেন্টকে আলাদা করা হয়। যেমন ৪ নাম্বার চ্যাপ্টারের অনেক জায়গাতেই মেথড কল দেখানো হয়েছে। নিচে আবার আমরা এক সেট উদাহরণ তৈরি করব এই চ্যাপ্টারের টপিক গুলো বোঝানোর জন্য,

```
// Food.h

#import

@interface Food : NSObject

@property (nonatomic) int price;

- (void)makeWithType:(NSString *) type withSize:(int) size;

@end
```

```
// Food.m

#import "Food.h"

@implementation Food

- (void)makeWithType:(NSString *)type withSize:(int)size {

    // Setting price property
    self.price = size*60;

    // Logging in the console
    NSLog(@"Making a %d inch %@ that costs %d Tk", size, type, self.price);
}

@end
```

```
// main.m

#import
#import "Food.h"

int main(int argc, const char * argv[])
{
    @autoreleasepool {

        Food *pizza = [[Food alloc] init];

        [pizza makeWithType:@"Mexican Pizza" withSize:6];

    }
    return 0;
}
```

উপরে খুব সহজ একটি উদাহরণ দেখানো হয়েছে যেখানে প্রথমে Food ক্লাসের ইন্টারফেসে একটি প্রোপার্টি price এবং একটি মেথড makeWithType:withSize: ডিক্লেয়ার করা হয়েছে এবং এদের ইমপ্লিমেন্টেশনটাও খুব সহজ যেখানে এই মেথড তার দ্বিতীয় আর্গুমেন্ট হিসেবে পাওয়া পিৎজা সাইজের সাথে একটা ভ্যালু ৬০ গুন করে price প্রোপার্টির ভ্যালু সেট করছে এবং শেষে কনসোলে একটি ম্যাসেজ প্রিন্ট করছে। ইতোমধ্যে হয়ত খেয়াল করেছেন একাধিক প্যারামিটার ওয়ালা মেথডকে নিচের মত করে কল করতে হয়,

```
[pizza makeWithType:@"Mexican Pizza" withSize:6];
```

অর্থাৎ- দ্বিতীয় এবং এর পরবর্তী প্যারামিটার গুলো, ইনিশিয়ালটির (প্রথম) পর পরই থাকে এবং প্রত্যেকটি প্যারামিটার তার আগেরটির সাথে একটি "স্পেস লেবেল:ভ্যালু" প্যাটার্ন মেইন্টেইন করে।

এই অঙ্কুরিত নিয়মটাই কিন্তু এদানিং বেশি হিউম্যান রিডেবল মনে করা হয়। অনেকদিন পর যদি হুট করে কোনদিন আপডেট অথবা বাগ ফিক্সিং এর উদ্দেশ্যে আগের করা প্রজেক্টটি আপনি যদি নতুন করে বুঝতে চান তখনই আসলে বোঝা যাবে এভাবে নেমিং কনভেনশন মেনে মেথডের নামকরণের সুবিধা।

নেস্টেড মেথড কলঃ নেস্টেড মেথড কলিং, অবজেক্টিভ সি এর একটি কমন প্যাটার্ন। একটি মেথডের রিটার্ন করা ভ্যালু অন্য মেথডে পাঠানোর স্বাভাবিক উপায়কেই নেস্টেড মেথড কল বলা হয়। অন্যান্য ল্যাঙ্গুয়েজের ডিউ থেকে থেকে দেখলে এটা একধরনের চেইনিং মেথড কল। উপরের উদাহরণেও কিন্তু একটা নেস্টেড মেথড কল হয়েছে। main.m ফাইলের ১২ নম্বার লাইনে,

```
Food *pizza = [[Food alloc] init];
```

এখানে [Food alloc] এর মাধ্যমে একবার alloc মেথড কল করে অবজেক্টের জন্য মেমোরি অ্যালোকেট করা হয়েছে এবং তারপরই এটার রিটার্ন ভ্যালুকে দিয়ে init মেথড কল করা হয়েছে।

প্রোটেক্টেড এবং প্রাইভেট মেথডঃ অবজেকটিভ সি তে কোন প্রোটেক্টেড অথবা প্রাইভেট এক্সেস মডিফায়ার নেই। এতে সকল মেথডই পাবলিক যদি সেগুলো ইন্টারফেস/এপিআই/পাবলিক লেয়ারে ডিক্লেয়ার করা হয়। তারপরও কোন মেথড এর ডিক্লেয়ারেশন হেডার ফাইলে না লিখে যদি শুধু ইমপ্লিমেন্টেশন ফাইলে লেখা হয় তাহলে মেথডটি প্রাইভেট মেথডের মত আচরণ করে। কারন, অন্যান্য অবজেক্ট বা সাবক্লাস সাধারণত .m বা ইমপ্লিমেন্টেশন ফাইলকে ইম্পোর্ট করে না আর তাই ওগুলো শুধুমাত্র ওই ক্লাসেরই অ্যাসেট হিসেবে থেকে যায়। একি কথা প্রোপার্টির জন্যও প্রযোজ্য।

নিচের ৩টি ফাইল মিলে এরকম একটি কেস তৈরি করা হয়েছে।

```
// Food.h

#import

@interface Food : NSObject

-(void)makeWithType:(NSString *) type withSize:(int) size;

@end
```

```
// Food.m

#import "Food.h"

@implementation Food {
    // The following variable is not directly accessible by any object of Food class
    int _price;
}

// The following method is not directly accessible by any object of Food class
- (int) calculatePrice: (int) size {

    // Setting the value of a private instance variable
    _price = size*60;

    return _price;
}

- (void)makeWithType:(NSString *)type withSize:(int)size {

    // Calling a kind of private method
    [self calculatePrice:size];

    // Logging in the console
    NSLog(@"Making a %d inch %@ that costs %d Tk", size, type, _price);
}

@end
```

```
// main.m

#import
#import "Food.h"

int main(int argc, const char * argv[])
{

    @autoreleasepool {

        Food *pizza = [[Food alloc] init];

        // [pizza calculatePrice:6]; // Is not accessible like this way

        [pizza makeWithType:@"Mexican Pizza" withSize:6];

    }
    return 0;
}
```

এখানে `_price` ভ্যারিয়েবলটি এবং `calculatePrice:` মেথডটি প্রাইভেট এর মত আচরন করে। অর্থাৎ উপরের উদাহরণটি ঠিক ঠাক মতোই কাজ করবে কিন্তু যদি `main.m` ফাইলের ১৩ নম্বার লাইনটি আনকমেন্ট করে প্রোগ্রামটি রান করতে চান তাহলে Xcode এরর ম্যাসেজ দিয়ে জানাবে যে `Food` ইন্টারফেসে `calculatePrice:` নামে কোন



মেথড নাই।

যার মানে দাডায়, এই

মেথডকে আপনি ওই ক্লাসের বাইরে থেকে ব্যবহার করতে পারছেন না। কিন্তু `calculatePrice:` মেথডটি ঠিকই আমরা ব্যবহার করতে পারছি `Food.m` ফাইলে অর্থাৎ ক্লাসের ভিতরে, যেমন ২৪ নম্বার লাইনে।

অন্যদিকে অবজেক্টিভ সি তে, প্রোটেক্টেড মেথডের বিকল্প এবং এর বিস্তারিত ব্যবহার পাওয়া যাবে ক্যাটাগরি সেকশনে। অবজেক্টিভ সি তে ক্যাটাগরি নিয়ে সামনেই পুরো একটা চ্যাপ্টার আসছে। তাই এখানে আর এটা নিয়ে আলোচনা করা হচ্ছে না।

সিলেক্টরঃ অবজেক্টিভ সি তে সিলেক্টর হচ্ছে কোন মেথডের নামের ইন্টারনাল রিপ্রেজেন্টেশন। এর মাধ্যমে আপনি একটি মেথডকে একটি স্বাধীন এলিমেন্ট হিসেবে এর সাথে ডিল করতে পারবেন। যেমন, একটি অবজেক্ট সাধারণত একটি অ্যাকশনকে ব্যবহার করে আর এ ক্ষেত্রে সিলেক্টর অবজেক্ট থেকে ওই অ্যাকশনকে আলাদা করে। এটাকে Target-Action ডিজাইন প্যাটার্ন এর ভিত্তি বলা যায়। আর সহজ ভাবে বলতে গেলে কোন মেথডকে ডাইনামিক্যালি ইনভোক/কল করার জন্য সিলেক্টরের দরকার পরে।

দুইটি পদ্ধতিতে একটি মেথড এর নামের সাপেক্ষে একটি সিলেক্টর পাওয়া যায়। `@selector()` ডিরেক্টিভ ব্যবহার করে অথবা `NSStringFromClass()` ফাংশনের সাহায্যে। দুইটি ক্ষেত্রেই এরা আপনার নতুন সিলেক্টর এর জন্য একটি ডাটা টাইপ রিটার্ন করে যার নাম SEL. অন্যান্য সাধারণ ডাটা টাইপ যেমন BOOL, int এর মতই এই SEL

কেও একটি ডাটা টাইপ মনে করা যেতে পারে।

নিচের মত করে Food ক্লাসের ইন্টারফেস, ইমপ্লিমেন্টেশন এবং main ফাইলের কোড আপডেট করুন। তারপর আমরা সিলেক্টর এর ব্যবহার আরও ভাল ভাবে উপলব্ধি করতে পারবো।

```
// Food.h

#import

@interface Food : NSObject

- (void)placeOrder;
- (void)makeOfType:(NSString *) type ofSize:(NSNumber *) size;

@end
```

```
// Food.m

#import "Food.h"

@implementation Food

- (void)placeOrder {
    NSLog(@"Order Received!");
}

- (void)makeOfType:(NSString *)type ofSize:(NSNumber *)size {

    NSLog(@"Making a %@ inch %@", size, type);
}

@end
```

```
// main.m

#import
#import "Food.h"

int main(int argc, const char * argv[])
{

    @autoreleasepool {

        Food *choice = [[Food alloc] init];

        SEL stepOne = NSSelectorFromString(@"placeOrder");
        SEL stepTwo = @selector(makeOfType:ofSize:);

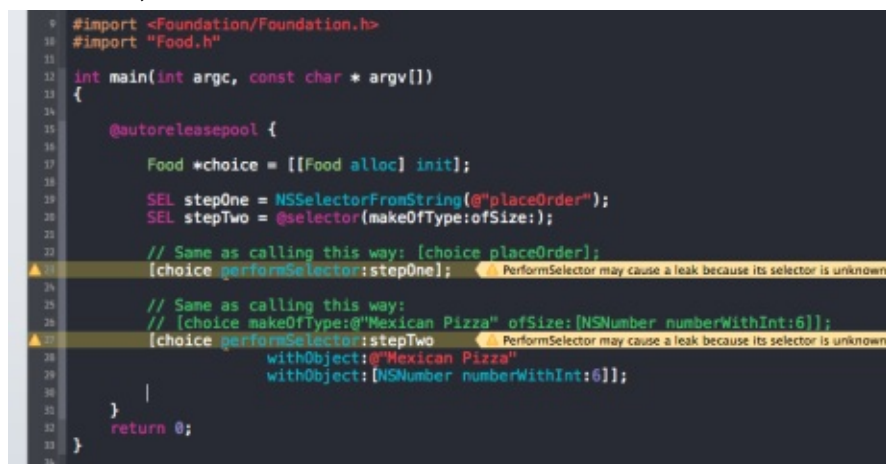
        #pragma clang diagnostic push
        #pragma clang diagnostic ignored "-Warc-performSelector-leaks"

        // Same as calling this way: [choice placeOrder];
        [choice performSelector:stepOne];

        // Same as calling this way:
        // [choice makeOfType:@"Mexican Pizza" ofSize:[NSNumber numberWithInt:6]];
        [choice performSelector:stepTwo
            withObject:@"Mexican Pizza"
            withObject:[NSNumber numberWithInt:6]];

        #pragma clang diagnostic pop
    }
    return 0;
}
```

এখানে খুব স্বাভাবিক ভাবেই ইন্টারফেস এবং ইমপ্লেমেন্টেশন ঠিক করা হয়েছে যে ব্যাপারে আপনি ইতোমধ্যে কয়েকবার ধারণা পেয়েছেন। এরপর main.m ফাইলে আমরা দুটি সিলেক্টর তৈরি করে নিয়েছি উপরে উল্লেখিত দুটি পদ্ধতি ব্যবহার করেই ১৩ এবং ১৪ নম্বার লাইনে। এর পর ১৬, ১৭ এবং ২৮ নম্বার লাইনের কোডগুলো আমাদের সিলেক্টর এর ব্যবহার সম্পর্কিত নয় বরং কম্পাইলারের কিছু ওয়ার্নিং বন্ধ করার জন্য। এভাবে সিলেক্টর ডিফাইন করলে কম্পাইলার বুঝে উঠতে পারে না যে, এই মেথড গুলোর রেজাল্ট দিয়ে কি করা হবে। তাই Xcode থেকে নিচের মত



```

10 #import <Foundation/Foundation.h>
11 #import "Food.h"
12
13 int main(int argc, const char * argv[])
14 {
15     @autoreleasepool {
16         Food *choice = [[Food alloc] init];
17         SEL stepOne = NSSelectorFromString(@"placeOrder");
18         SEL stepTwo = @selector(makeOfType:ofSize:);
19
20         // Same as calling this way: [choice placeOrder];
21         [choice performSelector:stepOne];
22
23         // Same as calling this way:
24         // [choice makeOfType:@"Mexican Pizza" ofSize:[NSNumber numberWithInt:6]];
25         [choice performSelector:stepTwo
26             withObject:@"Mexican Pizza"
27             withObject:[NSNumber numberWithInt:6]];
28     }
29     return 0;
30 }

```

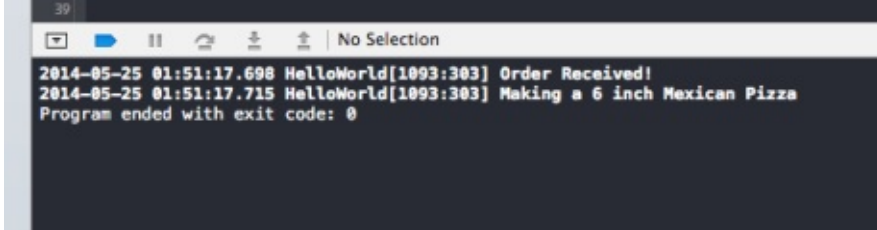
ওয়ার্নিং পেতে পারেন,

এই

ওয়ার্নিং এর ব্যপারে আরও বিস্তারিত [এখানে](#)

এরপর ২০ এবং ২৪ নাম্বার লাইনে আমরা performSelector: মেথড ব্যবহার করে আমাদের তৈরি করা সিলেক্টর গুলোকে এক্সিকিউট করেছি। ১৯ এবং ২২ নাম্বার লাইনে কমেন্ট এর মধ্যে বলা হয়েছে এর সমতুল্য ট্র্যাডিশনাল কল কেমন হতে পারতো। একাধিক প্যারামিটার ওয়ালা মেথডের ক্ষেত্রে withObject: ব্যবহার করে একের পর এক প্যারামিটার পাস করা হয়।

প্রোগ্রামটি রান করলে নিচের মত আউটপুট আসবে,

A screenshot of a terminal window with a dark background. The title bar shows '39' and 'No Selection'. The output text is as follows:

```
2014-05-25 01:51:17.698 HelloWorld[1093:303] Order Received!  
2014-05-25 01:51:17.715 HelloWorld[1093:303] Making a 6 inch Mexican Pizza  
Program ended with exit code: 0
```

পরের চ্যাপ্টারঃ এই চ্যাপ্টার পর্যন্ত অবজেক্টিভ সি এর ব্যাসিক ব্যপার গুলো সংক্ষেপে তুলে ধরা হয়েছে। পরবর্তী চ্যাপ্টার থেকে একটু অ্যাডভান্স বিষয় নিয়ে আলোচনা হবে। যেমন ৭ নাম্বার চ্যাপ্টারেই থাকবে প্রোটোকল (protocol) যার মাধ্যমে একটি এপিআই কে বিভিন্ন রকম এবং অনেক গুলো ক্লাস থেকে ব্যবহার করা যাবে।

Originally Posted [Here](#)

প্রোটোকল

ভূমিকাঃ অবজেক্টিভ সি তে আপনি প্রোটোকল নামের একধরনের ফিচার ডিফাইন করতে পারবেন যেখানে কিছু মেথড ডিক্লেয়ার করা যেতে পারে যে মেথডগুলো বিশেষ পরিস্থিতিতে ব্যবহার করা যায়। যে ক্লাস ওই প্রোটোকলকে মেনে চলবে সেই ক্লাসে ওই প্রোটোকলের মেথড গুলোকে ইমপ্লিমেন্ট করা হয়। আমরা নিচে যে উদাহরণ দেখবো সেখানে Food ক্লাসটি MakeProtocol কে মেনে চলবে, যে প্রোটোকলে processCompleted মেথডটি ডিক্লেয়ার করা আছে, যে মেথডের কাজ হচ্ছে খাবার তৈরি শেষ হয়ে গেলে ওই Food ক্লাসকে তা জানিয়ে দেয়া। অর্থাৎ, যেকোনো জায়গা থেকে Food ক্লাসের মধ্যে ইমপ্লিমেন্ট/ডিফাইন করা processCompleted মেথডকে কল করা। processCompleted মেথডটিকে অবশ্যই Food ক্লাসে ইমপ্লিমেন্ট করতে হবে। আর তাই এই ঘটনা অবশ্যই ঘটতে বাধ্য যে, Food ক্লাসের processCompleted মেথডটিকে এক্সিকিউট করা যাবেই।

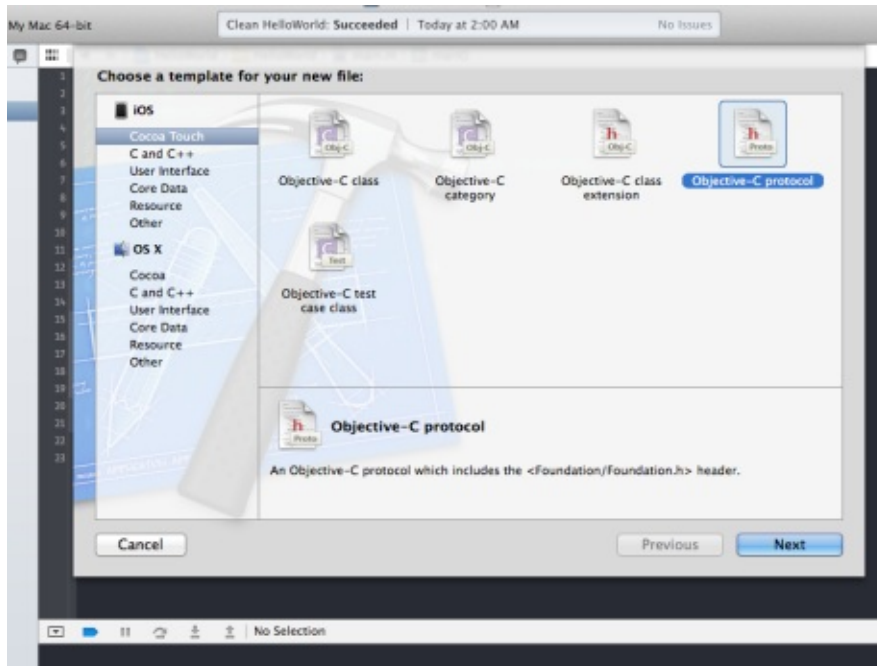
সিনট্যাক্সঃ

```
@protocol ProtocolName
@required
// list of required methods
@optional
// list of optional methods
@end
```

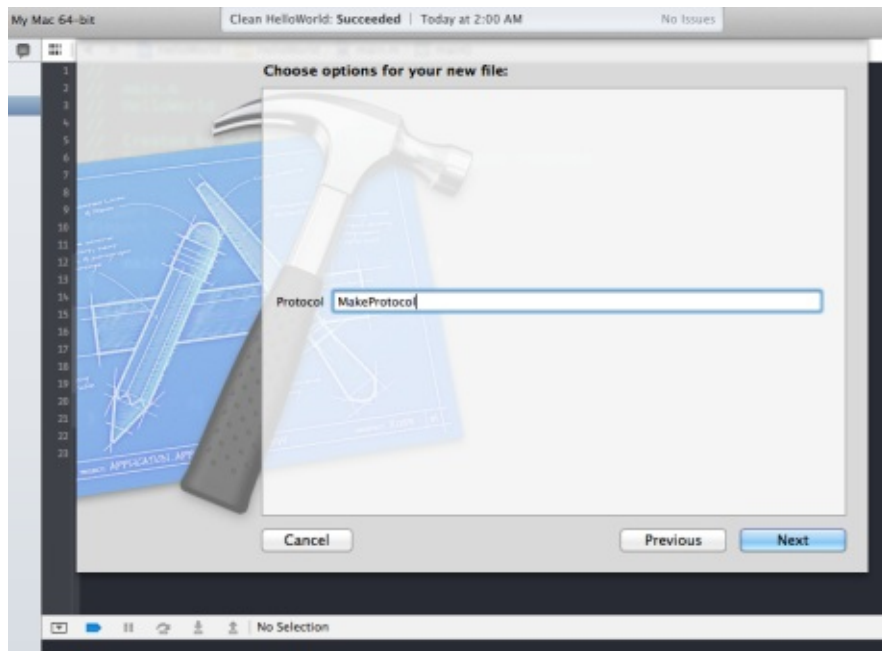
@required কিওয়ার্ডের নিচে থাকা মেথড/মেথডগুলোকে কে অবশ্যই সেই ক্লাসে ইমপ্লিমেন্টেড থাকতে হবে যে ক্লাস এই প্রোটোকলকে মেনে চলবে (conform)। আর @optional কিওয়ার্ডের নিচে থাকা মেথড/মেথডগুলোকে কে সেই ক্লাসে ইমপ্লিমেন্ট না করলেও চলবে। নিচের মত করে একটি ক্লাস যেকোনো একটি প্রোটোকলকে মেনে চলার ব্যপারে অঙ্গীকারাবদ্ধ হতে পারে।

```
@interface MyClass : NSObject
...
@end
```

প্রোটোকল তৈরিঃ যেকোনো প্রজেক্টে নিউ ফাইল হিসেবে প্রোটোকল ফাইলও যুক্ত করা যায়। Xcode এর File মেনু থেকে New->File ক্লিক করে নিচের মত করে একটি প্রোটোকল যুক্ত করে নিতে পারেন,



তারপর সেটির নাম ঠিক করে



দিতে পারেন এভাবে,

প্রোটোকল এর প্রয়োগঃ এখন আপনার প্রজেক্ট এর সাথে নিচের ফাইলগুলো মিলিয়ে একটি পরিষ্কার প্রজেক্ট সাজিয়ে ফেলুন। তারপর আমরা ওই প্রজেক্টে প্রোটোকল এর ব্যবহারটা বুঝতে চেষ্টা করব।

```
// MakeProtocol.h

#import

@protocol MakeProtocol

- (void)processCompleted;

@end
```

এই আমাদের বানানো processCompleted মেথড বিশিষ্ট একটি প্রোটোকল। যে ক্লাস এই প্রোটোকলকে মেনে চলবে তাকে অবশ্যই processCompleted মেথড ইমপ্লিমেন্ট করতে হবে।

```
// Make.h

#import

@interface Make : NSObject

@property (unsafe_unretained) id delegate;

- (void) startCooking;

@end
```

এটি হচ্ছে main এবং Food এর কর্মকাণ্ডের মধ্যবর্তী কিছু কাজের জন্য একটি ক্লাস এর ইন্টারফেস যেখানে একটি প্রোপারটি এবং একটি মেথড ডিক্লেয়ার করা আছে। delegate প্রোপারটিতে কোন ক্লাসের প্রতিনিধিত্ব সেট করা যায়। startCooking একটি সাধারণ মেথড যার বিস্তারিত একটু পরে Make এর ইমপ্লিমেন্টেশন থেকেও বোঝা যাবে।

```
// Make.m

#import "Make.h"
#import "MakeProtocol.h"

@implementation Make

- (void)startCooking{
    NSLog(@"Cooking Started!");

    // The following delegate property refers to
    // the Food class's object which was set from inside
    //// placeOrder
    // method of Food.

    // Then, a fact is also assured here that, the
    //// processCompleted
    // method will must be called from here which will
    // let the calling object know about its completion.

    [_delegate processCompleted];
}

@end
```

startCooking মেথডের শুরুতে আমরা একটি ম্যাসেজ প্রিন্ট করছি যে কুकिং শুরু হয়ে গেছে। তারপর আমরা এই ক্লাসের ডেলিগেট প্রোপার্টি ব্যবহার করে processCompleted ডেলিগেট মেথডকে কল করছি। এটা মূলত Food ক্লাসের মধ্যে থাকা processCompleted কেই কল করছে কারন একটু নিচেই আমরা দেখবো যে, Food ক্লাসের মধ্যে একটি মেথডের মাধ্যমে আমরা এই Make ক্লাসের ডেলিগেটটি সেট করছি অর্থাৎ Make ক্লাসের ডেলিগেট মূলত Food ক্লাসের অবজেক্ট হিসেবেই কাজ করছে।

```
// Food.h

#import
#import "MakeProtocol.h"

@interface Food : NSObject

- (void)placeOrder;

@end
```

উপরের কোড ব্লকে দেখতে পাচ্ছেন কিভাবে Food ক্লাসটি MakeProtocol নামের প্রোটোকলকে মেনে চলার ব্যাপারে নির্ধারিত হচ্ছে। সুপার ক্লাসের নামের পরে < MakeProtocol > এভাবে প্রোটোকল এর নাম ঠিক করে দেয়া যায়। যদি এই ক্লাসটি একাধিক প্রোটোকলকে মেনে চলতে চায় তাহলে < MakeProtocol, AnotherProtocol, ... > এভাবে প্রোটোকল লিস্ট ঠিক করে দেয়া যায়। এই ক্লাসে placeOrder নামের একটি সাধারণ মেথডও আছে যেখান থেকেই আসলে খাবারের অর্ডার নিয়ে কুकिং শুরু সহ বাকি কাজগুলো হয়।

```
// Food.m

#import "Food.h"
#import "Make.h"

@implementation Food

- (void)placeOrder{
    Make *make = [[Make alloc] init];
    make.delegate = self;
    [make startCooking];
}

-(void)processCompleted{
    NSLog(@"Cooking Process Completed & this method has been called!");
}

@end
```

Food এর ইমপ্লিমেন্টেশনে প্রথমত placeOrder মেথড এর কার্যক্রম ঠিক করে দেয়া হয়েছে যেখানে বলা হয়েছে যে, প্রথমে Make ক্লাস ইন্সট্যান্সিয়েট করে সেটির ডেলিগেট প্রোপার্টি সেট করা হচ্ছে self (Food) হিসেবে। তারপর Make ক্লাসের startCooking মেথডকে কল করা হচ্ছে। আর startCooking মেথড কি করে সেটাতো একটু উপরেই বলা হয়েছে। দ্বিতীয়ত, এখানেই Food ক্লাস বাধ্যগতভাবে processCompleted মেথডকে ইমপ্লিমেন্ট করছে। মনে করিয়ে দেই, এই মেথডটি কিন্তু Make ক্লাসের startCooking মেথডের মধ্যে থেকে কল হয় এবং আমরা বুঝতে পারি যে কুকিং কমপ্লিট হয়েছে আর এখানে সেই অনুযায়ী বাকি কাজ করি (এক্ষেত্রে কমপ্লিট হবার একটি ম্যাসেজ প্রিন্ট করছি)। যদি এখানে (Food ক্লাসে) প্রোটোকল মেথডটিকে ইমপ্লিমেন্ট না করতাম তাহলে Xcode



নিচের মত ওয়ার্নিং দিত,

```
// main.m

#import
#import "Food.h"

int main(int argc, const char * argv[])
{

    @autoreleasepool {

        Food *choice = [[Food alloc] init];
        [choice placeOrder];

    }
    return 0;
}
```

বুঝতেই পারছেন, পুরো প্রোগ্রামটির শুরু এখান (main()) থেকেই। প্রথমে Food ক্লাস ইন্সট্যান্সিয়েট করা হচ্ছে। তারপর এর placeOrder মেথড কল করে অর্ডার সাবমিট করা হচ্ছে এবং পরবর্তীতে কুकिং শুরু, শেষ এবং Food ক্লাসকে জানিয়ে দেয়ার কাজগুলো হচ্ছে যেসব উপর থেকে আমরা বর্ণনা করে এসেছি।

main থেকে শুরু করে *Food*, তারপর *Make*, মাঝখানে *MakeProtocol* এভাবে যদি আপনি নিজের মাথা দিয়ে কোড কম্পাইল করে করে উপরের দিকে আগান তাহলে আরেকবার পুরো উদাহরণটি বুঝতে উঠতে পারবেন বলে আশা করছি।

প্রোগ্রামটি রান করলে নিচের মত আউটপুট আসবে,

```
2014-06-02 20:01:47.165 HelloWorld[373:303] Cooking Started!
2014-06-02 20:01:47.167 HelloWorld[373:303] Cooking Process Completed & this method has been called!
Program ended with exit code: 0
```

iOS অ্যাপে প্রোটোকল এর প্রয়োগঃ রিয়াল লাইফ iOS অ্যাপ্লিকেশন ডেভেলপমেন্টে প্রোটোকল এর ব্যবহার খুব সাধারণভাবেই চলে আসে। যেমনঃ "application delegate" অবজেক্ট সেরকম একটি ব্যাপার যেটি মূলত ওই অ্যাপ রান হওয়া থেকে শুরু করে যতক্ষণ চলবে ততক্ষণ বিভিন্ন ইভেন্ট হ্যান্ডেল করে থাকে। রিয়াল অ্যাপ ডেভেলপ করার সময় নিচের কোড ব্লকটুকু আপনার নজরে চলে আসবে,

```
@interface YourAppDelegate : UIResponder
```

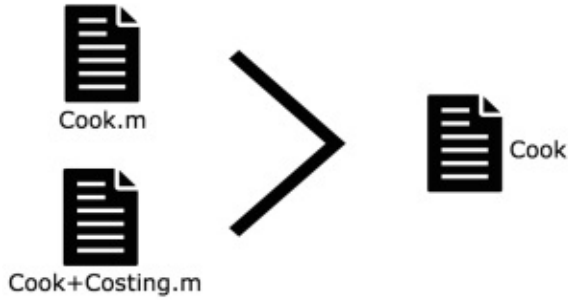
যতক্ষণ পর্যন্ত এটা (আপনার "application delegate" অবজেক্ট) UIApplicationDelegate এর মধ্যে ডিফাইন করা মেথডগুলোকে রেসপন্স করবে ততক্ষণ পর্যন্ত যেকোনো অবজেক্টকেই আপনার অ্যাপ্লিকেশন ডেলিগেট হিসেবে ব্যবহার করতে পারবেন। এতে করে অ্যাপ ইভেন্টগুলো নিয়ে কাজ করতে সুবিধা হবে।

পরের চ্যাপ্টারঃ এই চ্যাপ্টারে আমরা প্রোটোকল নিয়ে একটি উদাহরণ এর মাধ্যমে এর ব্যবহার মোটামুটি জানতে পেরেছি। যদিও আমাদের বইয়ে আরও বিস্তারিত বিশ্লেষণ থাকবে বলে আশা করি। পরের চ্যাপ্টারে খুব মজার এবং কাজের দুটি ফিচার অর্থাৎ অবজেক্টিভ সি এর ক্যাটাগরি (Categories) এবং ব্লক (Blocks) নিয়ে আলোচনা করা হবে।

Originally Posted [Here](#)

ক্যাটাগরি

ভূমিকাঃ সংক্ষেপে বলতে গেলে ক্যাটাগরি হচ্ছে একটি ক্লাসের ডেফিনেশনকে অনেক গুলো আলাদা আলাদা ফাইলে ভাগ করে নেয়া। অনেক বড় কোড বেজে কাজ করার সময় একটি ক্লাসের মডিউলারাইজেশন করাই এর উদ্দেশ্য। মডিউলারাইজেশন বলতে বোঝানো হচ্ছে যে, একটি বড় ক্লাসকে ছোট ছোট অনেক গুলো স্বাধীন অংশ হিসেবে তৈরি করার একটা প্যাটার্ন।



উপরের ছবিটিতে এটাই প্রকাশ করা হয়েছে যে, একাধিক ফাইল অর্থাৎ Cook.m এবং Cook+Costing.m মিলে Cook ক্লাসের পূর্ণাঙ্গ ইমপ্লিমেন্টেশন সম্পন্ন হয়েছে। নিচের উদাহরণের মাধ্যমে আমরা আরও পরিষ্কার ধারণা পাবো যেখানে, একটি ক্যাটাগরি ব্যবহার করে একটি বেজ ক্লাসকে এক্সটেন্ড করব অর্থাৎ বেজ ক্লাসের অ্যাসেট হিসেবে আরও কিছু মেথড যুক্ত করব কিন্তু বেজ ক্লাসটির সোর্স বা কোন কিছু পরিবর্তন না করেই।

উদাহরণ এর পটভূমি তৈরিঃ কথামত, ক্যাটাগরি ব্যবহার বুঝতে হলে আমাদের এক সেট বেজ ক্লাস দরকার পরবে। আর তাই নিচের মত করে একটি ব্লক প্রজেক্টে Cook ক্লাস তৈরি করে নিন যেটা এক্ষেত্রে আমাদের বেজ ক্লাস।

```
Cook.h

#import

@interface Cook : NSObject

@property (copy) NSString *item;

- (void)placeOrder;
- (void)startCooking;

@end
```



```

Cook.m

#import "Cook.h"

@implementation Cook

-(void) placeOrder {
    NSLog(@"Order placed for a %@", self.item);
}

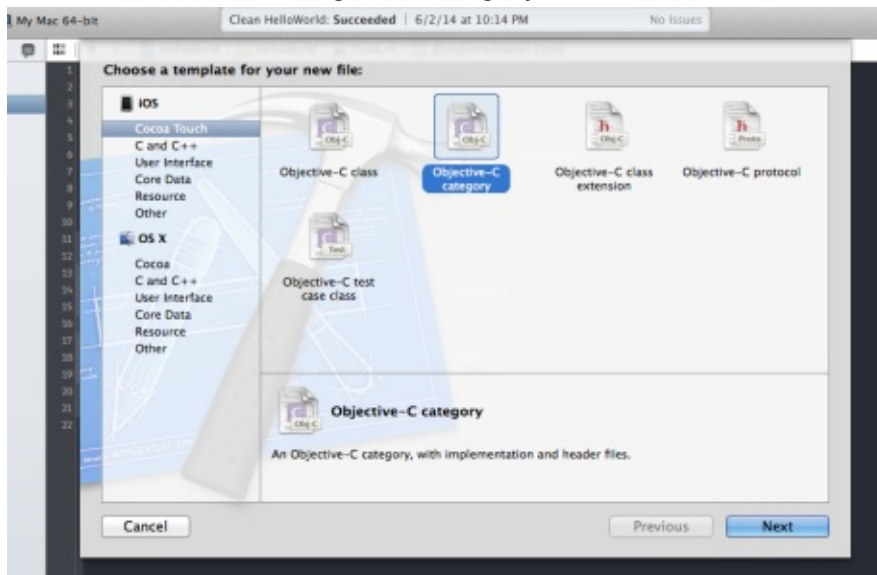
-(void) startCooking {
    NSLog(@"Started cooking %@", self.item);
}

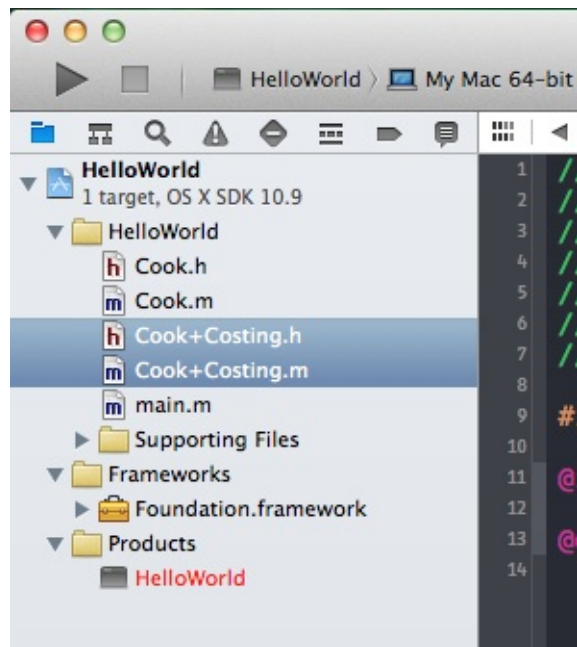
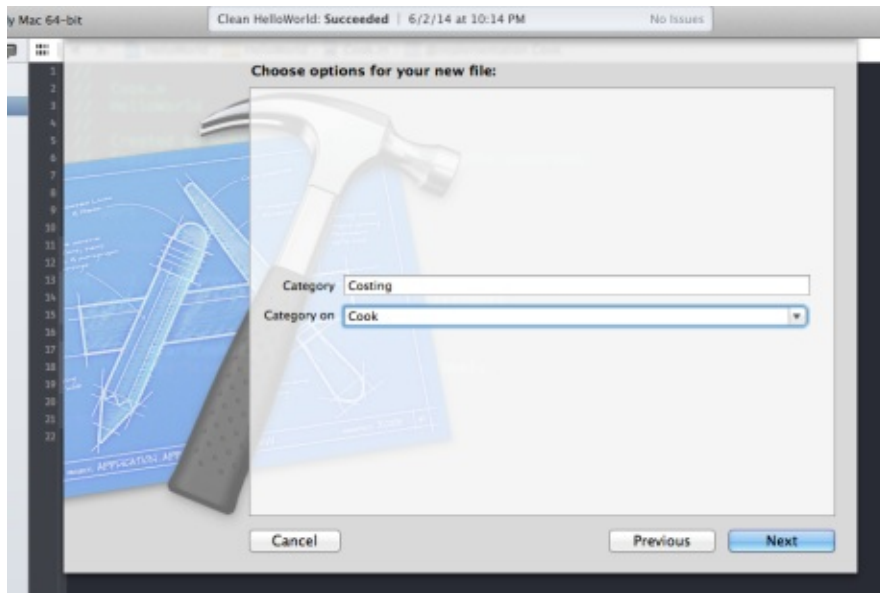
@end

```

আশা করি উপরের সাধারণ ক্লাস ডেফিনেশনটা বুঝতে পেরেছেন। এভাবে আগে আমরা অনেক বারই ক্লাস তৈরি করেছি যেখানে একটি ইন্টারফেস ফাইল এবং একটি ইমপ্লিমেন্টেশন ফাইল মিলে একটি পূর্ণ ক্লাস ডেফিনেশন সম্পন্ন হয়। এখন মনে করুন যে, আপনি রান্না (Cook) সম্পর্কিত আরও কিছু মেথড যুক্ত করতে চাচ্ছেন কিন্তু Cook ক্লাস এর কোন কিছু পরিবর্তন না করে বা সেটার গঠন নষ্ট না করে। এই ক্ষেত্রে আপনি ওই নতুন মেথড গুলো একটি ক্যাটাগরিতে যুক্ত করতে পারেন। ধরুন, আমরা চাই Cooking সম্পর্কিত খরচ এর হিসাবের একটি নতুন মেথড যুক্ত করতে।

ক্যাটাগরি তৈরিঃ ক্যাটাগরির ডেফিনেশন এবং ক্লাসের ডেফিনেশন মোটা মুঠি একি রকম অর্থাৎ এরও ইন্টারফেস এবং ইমপ্লিমেন্টেশন মিলেই সম্পূর্ণতা হয়। আমাদের প্রজেক্টে নতুন Cook+Costing নামের ক্যাটাগরি যুক্ত করতে হলে Xcode এর New File অপশন থেকে নিচের মত করে এই ক্যাটাগরিটি তৈরি ও প্রজেক্টে যুক্ত করুন। ক্যাটাগরির নাম দিচ্ছি Costing এবং Category On হিসেবে লিখতে/সিলেক্ট করতে হবে Cook.





তাহলে আমাদের প্রজেক্টে ফাইল স্ট্রাকচার হবে নিচের মত,

এখন একটু খেয়াল করলেই দেখবেন ক্যাটাগরির ইন্টারফেস ফাইল এবং সাধারণ ক্লাসের ইন্টারফেস ফাইল একি রকম দেখতে শুধু @interface এর পর ক্লাসের নাম এবং তারপর ব্রাকেটের মধ্যে ক্যাটাগরির নাম উল্লেখ করতে হয়। নিচে আমাদের Cook+Costing এর ইন্টারফেস,

```
Cook+Costing.h

#import "Cook.h"

@interface Cook (Costing)

- (BOOL)isFreeOfferOngoing;
- (void)calculatePrice;

@end
```

রানটাইমে এই মেথড দুটো Cook ক্লাসের অংশ হিসেবেই কাজ করে। যদিও এগুলো আলাদা একটা ফাইলে ডিফাইন করা হল তবুও এগুলোকে এমন ভাবে এক্সেস করা যাবে যেন মনে হবে এগুলো Cook ক্লাসের মেথড। আর আগেই বলা হয়েছে ক্যাটাগরির ইমপ্লিমেন্টেশনও লাগবে (সাধারণ ক্লাসের মতই)। তাই নিচের মত করে Cook+Costing.m ফাইলটি পরিবর্তন করুন,

```
Cook+Costing.m

#import "Cook+Costing.h"

@implementation Cook (Costing)

-(BOOL) isFreeOfferOngoing {
    return NO;
}

-(void)calculatePrice {
    NSLog(@"Price of %@ would be 350 tk.", self.item);
}

@end
```

ব্যবহারঃ যেকোনো ফাইল কোন একটি ক্যাটাগরির যেকোনো একটি API (ইন্টারফেস লেয়ার) ব্যবহার করতে চাইলে তাকে অবশ্যই ওই ক্যাটাগরির হেডার ফাইলে যুক্ত করে নিতে হবে। অর্থাৎ নিচের মত করে main.m ফাইলে প্রথমে আমাদেরকে Cook ক্লাসের হেডার এবং তারপরে Cook+Costing ক্যাটাগরির হেডার যুক্ত করতে নিতে হবে,

```
main.m

#import
#import "Cook.h"
#import "Cook+Costing.h"

int main(int argc, const char * argv[])
{

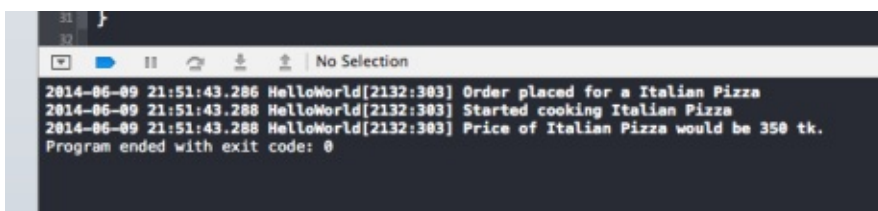
    @autoreleasepool {
        Cook *forMe = [[Cook alloc] init];
        forMe.item = @"Italian Pizza";

        // "Standard" functionality from Cook.h
        [forMe placeOrder];
        [forMe startCooking];

        // Additional methods from Cook+Costing.h
        if (![forMe isFreeOfferOngoing]) {
            [forMe calculatePrice];
        }

    }
    return 0;
}
```

উপরে খেয়াল করুন, খুব সুন্দর ভাবে Cook ক্লাসের ফাংশনালিটি গুলো আমরা এক্সেস করতে পারছি এবং সাথে ওই ক্লাসের অবজেক্ট বা ইন্সট্যান্স দিয়েই কিন্তু Costing ক্যাটাগরির মাধ্যমে Cooking এর জন্য অতিরিক্ত প্রয়োজনীয় মেথড যেগুলো নতুন ক্যাটাগরি হিসেবে ডিফাইন করা হয়েছে সেগুলোও এক্সেস করতে পারছি ১৯-২১ নাম্বার লাইনে। প্রোজেক্টটি রান করালে নিচের মত আউটপুট আসবে।



```
2014-06-09 21:51:43.286 HelloWorld[2132:303] Order placed for a Italian Pizza
2014-06-09 21:51:43.288 HelloWorld[2132:303] Started cooking Italian Pizza
2014-06-09 21:51:43.288 HelloWorld[2132:303] Price of Italian Pizza would be 350 tk.
Program ended with exit code: 0
```

প্রোটেক্টেড মেথড হিসেবে ক্যাটাগরিঃ আশা করি এই বিষয়ে আমাদের সম্ভাব্য কাণ্ডজে বইয়ে বিস্তারিত আলোচনা থাকবে।

এক্সটেনশনঃ আশা করি এই বিষয়ে আমাদের সম্ভাব্য কাণ্ডজে বইয়ে বিস্তারিত আলোচনা থাকবে।

Originally Posted [Here](#)

ব্লক ও এর ব্যবহার

ভূমিকাঃ সহজভাবে বলতে গেলে ব্লক হচ্ছে অবজেক্টিভ-সি এর anonymous function. Google ডেফিনেশন অনুযায়ী অ্যানোনিমাস ফাংশন হচ্ছে- *An anonymous function is a function that is not stored in a program file, but is associated with a variable whose data type is function_handle.* Anonymous functions can accept inputs and return outputs, just as standard functions do. However, they can contain only a single executable statement. এর মাধ্যমে আপনি বিভিন্ন অবজেক্টের মধ্যে ইচ্ছামত স্টেটমেন্ট (Statement) পাস তথা আদান-প্রদান করতে পারবেন যেমনভাবে সাধারণ ডাটা আদান প্রদান করে থাকেন। উপরন্তু ব্লককে ক্লোজার (Closure) হিসেবে ব্যবহার করা হয় যাতে করে তার পক্ষে তার আসে পাশের ডাটা/অবস্থা নিয়ে কাজ করাও সম্ভব হয়। আস্তে আস্তে আমরা এ ব্যাপারে বিস্তারিত জানতে চেষ্টা করব।

সিনট্যাক্সঃ ডিক্লেয়ারেশন-

```
returntype (^blockName) (argumentType);
```

ইমপ্লিমেন্টেশন-

```
returntype (^blockName) (argumentType) = ^{
    // Statements
};
```

এখানে "=" চিহ্নের বাম পাশের অংশটি হচ্ছে একটি ব্লক ডেরিয়েবল এবং ডান পাশের অংশ হচ্ছে ব্লকটি।

উদাহরণ-

```
void (^simpleBlock) (void) = ^{
    NSLog(@"This is a block that does not return anything and also has no argument!");
};
```

উপরের এই ব্লককে আমরা `simpleBlock();` এভাবে কল করতে পারি।

ব্লক (Block) তৈরিঃ ব্লক তৈরির ব্যাপারটা মোটা মুটি ফাংশন তৈরির মতই। যেভাবে আপনি একটি ফাংশন ডিক্লেয়ার করে থাকেন ঠিক সেভাবেই একটি ব্লক ডেরিয়েবল ডিক্লেয়ার করতে পারেন আবার যেভাবে একটি ফাংশন এর ইমপ্লিমেন্ট করেন সেভাবে একটি ব্লককে ডিফাইন করতে পারেন। এমনকি যেভাবে একটি ফাংশনকে কল করেন ঠিক সেভাবেই একটি ব্লককেও কল করতে পারেন। নিচে একটি সহজ উদাহরণ দেখানো হল,

```
// main.m

#import

int main(int argc, const char * argv[])
{
    @autoreleasepool {
        // Declare and define the isPlaceOpen block
        BOOL (^isPlaceOpen)(void) = ^ {
            return YES;
        };

        // Use the block
        NSLog(@"Open state of the place is: %hhd ", isPlaceOpen());
    }
    return 0;
}
```

"^" চিহ্নটি ব্যবহার করে isPlaceOpen কে একটি ব্লক হিসেবে চিহ্নিত করা হয়। এটাকে অবজেক্টিভ-সি এর "*" পয়েন্টার চিহ্নের মতই মনে করতে পারেন অর্থাৎ শুধুমাত্র ডিক্লেয়ার করার সময় এটা গুরুত্বপূর্ণ কিন্তু এর পরে এটাকে সাধারণ ডেরিয়েবলের মতই মনে করে ব্যবহার করতে পারেন। নিচে আরেকটি উদাহরণ দেখি যেখানে আমাদের ব্লকটি দুইটি double টাইপের প্যারামিটারও গ্রহণ করে এবং সেগুলো ব্যবহার করে অল্প কিছু হিসাব করার পর একটি double টাইপ ডাটা রিটার্ন করে।

```
// main.m

#import

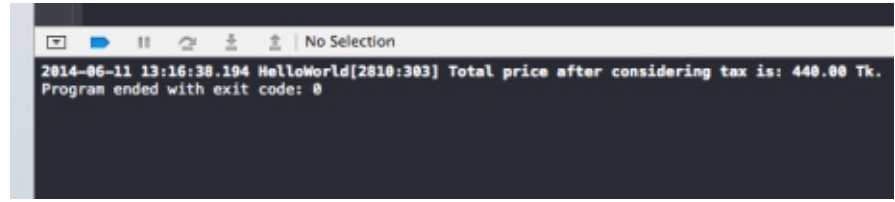
int main(int argc, const char * argv[])
{
    @autoreleasepool {
        // Declare the block variable
        double (^priceAfterTax)(double rawPrice, double totalTax);

        // Create and assign the block to the variable
        priceAfterTax = ^(double price, double tax) {
            return price + tax;
        };

        // Call the block
        double total = priceAfterTax(400, 40);

        NSLog(@"Total price after considering tax is: " @"%.2f Tk.", total);
    }
    return 0;
}
```

খেয়াল করুন, প্রথমে আমরা একটি ব্লক ভেরিয়েবল ডিক্লেয়ার করেছি তারপর আসল ব্লকটিকে সেই ভেরিয়েবলে অ্যাসাইন করেছি এবং তারপর ভেরিয়েবলের নাম ধরে সেই ব্লককে ব্যবহার বা কল করেছি। প্রোগ্রামটি রান করলে



নিচের মত আউটপুট আসবে।

ক্লোজার (Closure) :: শুরুতেই বলা হয়েছে যে, সাধারণভাবে একটি ব্লক, ক্লোজার হিসেবেও ইমপ্লিমেন্টেড হয়। যেমন নিচের উদাহরণটি দেখে আমরা বিশ্লেষণ করতে পারি এখানে ক্লোজার ফিচার কোথায়,

```
// main.m

#import

int main(int argc, const char * argv[])
{
    @autoreleasepool {
        NSString *foodGenre = @"Italian";

        NSString *(^getFullItemName)(NSString *) = ^(NSString *itemName) {
            return [foodGenre stringByAppendingFormat:@" %@", itemName];
        };

        NSLog(@"%@", getFullItemName(@"Pizza"));    // Honda Accord
    }
    return 0;
}
```

আবারও বলি, ফাংশনের মতই একটি ব্লকের মধ্যে থেকে আপনি সেটার লোকাল ভেরিয়েবলগুলো, তার কাছে যে প্যারামিটারগুলো পাস করা হয়েছে সেগুলো এবং যেকোনো গ্লোবাল ভেরিয়েবল অ্যাক্সেস করতে পারবেন। কিন্তু ক্লোজার হিসেবে ব্লকের ব্যবহারের ফলে আপনি কিছু নন-লোকাল ভেরিয়েবলকেও ব্লকের মধ্যে থেকে অ্যাক্সেস করতে পারবেন। নন-লোকাল ভেরিয়েবল হচ্ছে ব্লকের বাইরের কোন ভেরিয়েবল কিন্তু ব্লকের Lexical Scope এর আওতাভুক্ত। উপরের উদাহরণে foodGenre সেরকম একটি নন-লোকাল ভেরিয়েবল যেটাকে getFullItemName ব্লকের মধ্যে থেকেও ব্যবহার করা গেছে।

মেথড প্যারামিটার হিসেবে ব্লক ব্যবহারঃ একটি পূর্ণ ব্লককে যেকোনো মেথডের প্যারামিটার হিসেবেও ব্যবহার করা যায়। ওই ব্লক যে কাজটা করে মূলত সেই কাজটিকেই একটা প্যাকেজ সরূপ একটি প্যারামিটার হিসেবে যেকোনো মেথডের কাছে দেয়া যায়। নিচে সেরকম একটা উদাহরণ দেখবো আমরা। প্রথমে আপনার প্রজেক্টকে এমনভাবে সাজিয়ে নিন যাতে সেখানে Food.h, Food.m এবং main.m এই ৩টি ফাইল থাকে নিচের মত করে,

```
// Food.h

#import

@interface Food : NSObject

- (void)performActionWithCompletion: (void (^) ()) completionBlock;

- (void)calculatePriceWithTax: (double (^) (double price)) taxCalculatorBlock;

@end
```

```
// Food.m

#import "Food.h"

@implementation Food

- (void)performActionWithCompletion:(void (^) ()) completionBlock {

    NSLog(@"Started cooking...");
    completionBlock();
}

- (void)calculatePriceWithTax: (double (^) (double price)) taxCalculatorBlock {
    NSLog(@"Total price is: %f", taxCalculatorBlock(400));
}

@end
```



```
// main.m

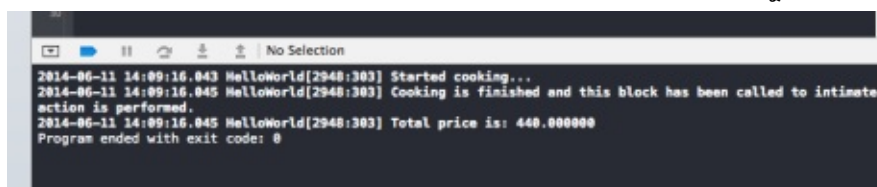
#import
#import "Food.h"

int main(int argc, const char * argv[])
{
    @autoreleasepool {
        Food *process = [[Food alloc] init];

        [process performActionWithCompletion: ^{
            NSLog(@"Cooking is finished and this block has been called to intimate action");
        }];

        [process calculatePriceWithTax: ^(double totalPrice) {
            return totalPrice + 40;
        }];
    }
    return 0;
}
```

উপরের কোড গুলো দেখুন এবং বোঝার চেষ্টা করুন, প্রথমেই আমরা Food ক্লাসের ইন্টারফেসে দুইটি মেথড ডিক্লেয়ার করেছি যে দুটো মেথড প্যারামিটার হিসেবে দুই ধরনের দুইটি ব্লক গ্রহণ করে। প্রথম মেথডটির নাম performActionWithCompletion: এবং এটি এমন একটি ব্লককে প্যারামিটার হিসেবে গ্রহণ করে যে ব্লকটির কোন কিছু রিটার্ন করে না এবং যার কোন আর্গুমেন্টও নেই। কিন্তু দ্বিতীয় মেথডটি একটি ব্লককে তার প্যারামিটার হিসেবে গ্রহণ করে যে ব্লক একটি double টাইপ ডাটা রিটার্ন করে এবং যার একটি আর্গুমেন্টও আছে। এখন এই দুইটি মেথড এর ইমপ্লিমেন্টেশন লিখেছি Food.m ফাইলে। অর্থাৎ এই দুইটি মেথড কি কাজ করবে তার ডেফিনেশন। প্রথম মেথডটি শুরুতেই একটি ম্যাসেজ প্রিন্ট করবে এবং তারপর ওর কাছে প্যারামিটার হিসেবে আসা ব্লকটিকে কল করবে। তো, এক্ষেত্রে কি ঘটবে? ওই completionBlock নামক ব্লকের মধ্যে যা করতে বলা হয়েছে তাই ঘটবে। অর্থাৎ ওখানে লিখে রাখা আরও একটি ম্যাসেজ প্রিন্ট হবে। দ্বিতীয় মেথড এর কাজ হচ্ছে, তার কাছে আসা ব্লকটিকে কল করবে। কিন্তু যেহেতু ওই ব্লকটির প্যারামিটার আছে তাই ওকে কল করার সময় একটি ড্যালুও পাস করে দিবে। আর ব্লক কল করা মানে কি? ওই ব্লকের মধ্যে যা করতে বলা হয়েছে তাই করবে। এক্ষেত্রে ওই ব্লকটি একটি ডাটা রিটার্ন করে এবং আসলে calculatePriceWithTax: মেথডটি সেই রিটার্ন করা ডাটাকেই নিজের মধ্যে থেকে প্রিন্ট করে। main.m ফাইলে আমরা উপরে উল্লেখিত মেথড দুইটি কল করেছি এবং তাদের প্যারামিটার হিসেবে দুইটি ভিন্ন ব্লককে পাঠিয়ে দিয়েছি। প্রোগ্রামট রান করলে নিচের মত আউটপুট আসবে,



```
2014-06-11 14:09:16.043 HelloWorld[2948:303] Started cooking...
2014-06-11 14:09:16.045 HelloWorld[2948:303] Cooking is finished and this block has been called to intimate
action is performed.
2014-06-11 14:09:16.045 HelloWorld[2948:303] Total price is: 440.000000
Program ended with exit code: 0
```

ব্লক টাইপ ডিফাইন করাঃ typedef এর মাধ্যমে একটি ব্লক টাইপ তৈরি করা যায় যাতে করে ব্লক ব্যবহারের সময় এর সিনট্যাক্স এলোমেলো না হয়ে যায় এবং ওই ব্লকটিকে আরও সহজ এবং সংক্ষেপ নামে ব্যবহার করা যায়। এ ব্যাপারে এবং উপরের টপিক গুলোর আরও বিস্তারিত থাকবে আমাদের সম্ভাব্য কাণ্ডজে বইয়ে। বই এর আপডেট পেতে লাইক

দিয়ে রাখতে পারেন [এখানে](#)।

Originally Posted [Here](#)

এক্সেপশন ও এরর হ্যান্ডেলিং

ভূমিকাঃ অ্যাপ্লিকেশন ডেভেলপমেন্ট এর সময় যে দুটি প্রধান সমস্যার কথা ডেভেলপার/প্রোগ্রামারদের ভাবতে হয় সেগুলো হল এক্সেপশন ও এরর। iOS ও OS X এ অ্যাপ্লিকেশন চলার সময়ও এক্সেপশন ও এরর ঘটতে পারে।

প্রোগ্রামার লেভেলের বাগ গুলো কে এক্সেপশন বলা হয়। উদাহরণস্বরূপ: যখন কোডের মাধ্যমে কোন অ্যারে (Array) এর ননএক্সিস্টিং ইলিমেন্টকে এক্সেস করার চেষ্টা করা হয় তখন সিস্টেম এক্সেপশন থ্রো করে। অ্যাপ্লিকেশন চলার সময় যদি কোন কারনে এরকম একটা বিষয় উদ্ভূত হয় তাহলে ডেভেলপারকে সেটা সম্পর্কে তথ্য দেয়ার জন্যই প্রোগ্রামিং ল্যান্সুয়েজগুলোতে এক্সেপশন ফিচার ডিজাইন করা হয়েছে। প্রোগ্রামের মধ্যেই প্রোগ্রামারকে এরকম এক্সেপশনাল কেস হ্যান্ডেল করার জন্য প্রয়োজনীয় কোড লিখতে হয় যাকে বলে এক্সেপশন হ্যান্ডেলিং।



এক্সেপশন ডেভেলপার কে জানাও।

অন্যদিকে ইউজার লেভেলে যদি কোন অনাকাঙ্ক্ষিত ঘটনা ঘটে সেগুলোকে এরর বলা হয়। যেমন, যদি আপনার অ্যাপ্লিকেশনটির কাজ হয় যে, ডিভাইসের GPS সার্ভিস ব্যবহার করে আসেপাশের কিছু লোকেশন দেখাবে। কিন্তু ইউজার তার ডিভাইসের GPS সেটিং ডিজ্যাবল করে রেখেছে। তখন অ্যাপটা চলার সময় ইউজারকে একটা ম্যাসেজ দেখানো উচিত যে, তার GPS অফ আছে। এরকম অবস্থায় ইউজারকে কিছুই না জানিয়ে অ্যাপ্লিকেশন এর ক্র্যাশ হওয়া উচিত হবে না। তাই এধরনের কন্ডিশনগুলো যখন ঘটবে তখন ম্যানুয়ালী চেক করে সঠিক তথ্যটা বা এরর ম্যাসেজটা ইউজারকে দেওয়াই হচ্ছে এরর হ্যান্ডেলিং।



ওয়ার্ক এরর ইউজারকে জানাও এবং সম্ভব হলে অ্যাপ রান করতেই থাকো।

কিন্তু মনে রাখবেন, কোডিং এরর আর উপরে উল্লেখিত এররটা কিন্তু এক জিনিস না। যেমন, আপনি আপনার প্রোগ্রামের মধ্যে একটা ক্লাস ইনস্ট্যান্সিয়েট করে সেটার কিছু মেথড এক্সেস করতে চাচ্ছেন কিন্তু শুরুতে ওই ক্লাসের হেডার ফাইলকে ইনক্লুড করেন নাই। তখন কিন্তু কম্পাইলার আপনাকে এরর দেবে এবং প্রোগ্রাম রান-ই করবে না। কম্পাইলারকে ডিজাইন করা হয়েছে এরকম কোডিং এরর সম্পর্কে ডেভেলপারকে জানাতে।



কোডিং এর ডেভেলপারকে জানিয়ে থেমে যাও

এক্সেপশন (**Exceptions**) : অবজেকটিভ-সি তে NSError ক্লাস দিয়ে এক্সেপশনকে রিপ্রেজেন্ট করা হয়। যখন কোন এক্সেপশন ঘটে তখন এই এক্সেপশনের যাবতীয় তথ্য NSError ক্লাসের ইনস্ট্যান্স থেকেই পাওয়া যায়। NSError ক্লাসের ৩টি প্রধান প্রোপার্টি নিচে দেওয়া হলঃ

- **name** : এক্সেপশনের নাম (ডাটাইপ: NSString) যা সকল এক্সেপশনের জন্য আলাদা
- **reason** : এক্সেপশনের সহজবোধ্য বর্ণনা (ডাটাইপ : NSString)
- **userInfo** : NSDictionary টাইপের অবজেক্ট। এই অবজেক্টে কি-ভ্যালু রিলেশনশিপ দিয়ে এক্সেপশনের অতিরিক্ত তথ্য থাকে

এক্সেপশন হ্যান্ডেলিং (**Exception Handling**) : নিচের মত করে একটি নতুন ফ্রেশ প্রজেক্টের main.m ফাইলটি এডিট করে ফেলিঃ

```
// main.m

#import

int main(int argc, const char * argv[]) {
    @autoreleasepool {

        // menuList of the restaurant
        NSArray *menuList = @[@"Vegetable Rice", @"Chicken Toast", @"Thai Soup"];

        NSLog(@"1st Item Name is %@", menuList[0]); // Vegetable Rice
        NSLog(@"2nd Item Name is %@", menuList[1]); // Chicken Toast
        NSLog(@"3rd Item Name is %@", menuList[2]); // Thai Soup
        NSLog(@"3rd Item Name is %@", menuList[3]); // throughs exception
    }
    return 0;
}
```

উপরের প্রোগ্রামটিতে প্রথমেই NSArray টাইপ একটি অ্যারে লিস্ট নেওয়া হয়েছে যাতে ৩ টি আইটেম আছে। আমরা জানি যে অবজেকটিভ সি এবং প্রায় সকল প্রোগ্রামিং ল্যাঙ্গুয়েজেই ০ থেকে ইনডেক্সিং শুরু হয়। সুতরাং menuList অ্যারেতে ০ থেকে ২ এই তিনটি ইনডেক্স রয়েছে। কিন্তু প্রোগ্রামটিতে ০ থেকে ৩ পর্যন্ত ইনডেক্স এক্সেস করা হয়েছে। প্রোগ্রামটিতে যদিও কোন সিন্ট্যাক্স এরর নেই তবুও এই প্রোগ্রামটি রান করানো হলে কনসোলে নিচের মত একটি মেসেজ ডিসপ্লে হবে।

```

2014-06-05 23:14:01.320 restaurant[598:303] *** Terminating app due to uncaught exception
*** First throw call stack:
(
  0   CoreFoundation                      0x00007fff9366641c __exceptionPreprocess + 17
  1   libobjc.A.dylib                    0x00007fff8e791e75 objc_exception_throw + 43
  2   CoreFoundation                      0x00007fff935521df -[__NSArrayI objectAtIndex:]
  3   restaurant                          0x00000000100001798 main + 376
  4   libdyld.dylib                      0x00007fff8a1405fd start + 1
)
libc++abi.dylib: terminating with uncaught exception of type NSException
(lldb)

```

প্রোগ্রামটি রান করলে এই এক্সেপশনটি থ্রো হয় এবং তারপর প্রোগ্রামটি ক্র্যাশ করে। কিভাবে হল? menuList এ তিনটি খাবারের নাম আছে যাদের ইনডেক্স যথাক্রমে ০, ১, ২। প্রোগ্রামটি কম্পাইল করলে কোন রকম এরর বা অন্য কোন ধরনের অস্বাভাবিক ঘটনা ঘটে না। কিন্তু কম্পাইলেশনের পর রান করলে শেষলাইনটি এক্সিকিউশনের সময় menuList[3] দিয়ে menuList এর ৩ নং ইনডেক্সের ভ্যালু এক্সেস করা হয়, তখনই এক্সেপশন (ব্যতিক্রম) ঘটে এবং প্রোগ্রামটি ক্র্যাশ করে কারণ menuList অ্যারেতে ০-২ এই তিনটি মাত্র ইনডেক্সই আছে।

এভাবে অনেক কারনেই অ্যাপ্লিকেশনটি রান করানোর পর এক্সেপশন হয়ে ক্র্যাশ হতে পারে যা মোটেই কাম্য নয়। তাই প্রোগ্রামেই যেখানে যেখানে এধরনের এক্সেপশন হওয়ার সম্ভাবনা থাকে সেসব জায়গায় এক্সেপশন হ্যান্ডেল করতে হয়।

অন্যান্য হাই লেভেল প্রোগ্রামিং ল্যাঙ্গুয়েজের মতই অবজেকটিভ-সি তেও try-catch-finally প্যাটার্নে এক্সেপশন হ্যান্ডেল করা হয়। প্রথমেই কোডের যেসব লাইনে এক্সেপশন হতে পারে সেগুলোকে @try ব্লকের ভেতরে লেখা হয়। যদি কোন এক্সেপশন ঘটে তাহলে @catch ব্লকে এক্সেপশনের ধরন জেনে এক্সেপশনটিকে হ্যান্ডেল করতে হয়। অবশেষে @finally ব্লকের কোড গুলো এক্সিকিউট করা হয়। এক্সেপশন হোক আর না হোক @finally ব্লক এক্সিকিউট হবেই।

```
// main.m

#import

int main(int argc, const char * argv[]) {
    @autoreleasepool {

        // menuList of the restaurant
        NSArray *menuList = @[@"Vegetable Rice", @"Chicken Toast", @"Thai Soup"];

        @try{
            // write codes where exception can be thrown
            NSLog(@"4th Item Name is %@", menuList[3]);
        }
        @catch(NSException *theException){
            // exception occurred. Lets handle this exception
            NSLog(@"An Exception named %@ has occurred@", theException.name);
            NSLog(@"Some more details about the exception : %@", theException.reason);
        }
        @finally{
            NSLog(@"Continues without crashing the application");
        }

        NSLog(@"This is being displayed. That means this application didn't crash.");
    }
    return 0;
}
```

উপরের main.m কে রান করলে @try ব্লকে menuList এর ৪র্থ আইটেম এক্সেস করার সাথে সাথেই এক্সেপশন থ্রো হয়। @catch ব্লকে NSException এর *theException এ এক্সেপশনটি অ্যাসাইন হয়। NSException ক্লাসের অবজেক্ট theException এর প্রোপার্টিগুলো name, reason, userInfo তে এক্সেপশনটি সম্পর্কে তথ্য পাওয়া যাচ্ছে।

রিয়েল লাইফ অ্যাপ্লিকেশনগুলোতে @catch ব্লকে সাধারণত এক্সেপশনের name ও reason থেকে তথ্য নিয়ে সমস্যার সমাধান করে অথবা ইউজারকে সহজবোধ্য এরর মেসেজ দিয়ে প্রোগ্রামটি কে নরমালি এক্সিট/কন্টিনিউ করা হয়।

অবজেকটিভ-সি এর এক্সেপশন হ্যান্ডেলিং খুব বেশী ইফিশিয়েন্ট না হওয়ায় @try/@catch ব্লক ব্যবহার না করে যদি সাধারণ if স্টেটমেন্ট দিয়ে চেক করে সমস্যা বাইপাস করা যায় তাহলে সেটা করাই ভাল। একান্তই যদি দরকার পড়ে সেক্ষেত্রে উপরের মত @catch ব্লকে ইউজার কে এক্সেপশনের সহজবোধ্য এরর মেসেজ দিয়ে প্রোগ্রামটি এক্সিট করে দেওয়া যেতে পারে।

```
// main.m

#import

int main(int argc, const char * argv[]) {
    @autoreleasepool {

        // menuList of the restaurant
        NSArray *menuList = @[@"Vegetable Rice", @"Chicken Toast", @"Thai Soup"];

        int selectedIndex = 3;

        if(selectedIndex < [menuList count]){
            NSLog(@"4th Item Name is %@", menuList[selectedIndex]);
        }
        else
        {
            //Exception is avoided
        }
    }
    return 0;
}
```

এই প্রোগ্রামটিতে শুরুতেই selectedIndex এর ভ্যালু চেক করে নেওয়া হচ্ছে। যদি selectedIndex টি menuList এর count এর চেয়ে ছোট হয় তাহলে ওই ইনডেক্সের ভ্যালু এক্সেস করা হচ্ছে। selectedIndex এর ভ্যালু যদি menuList এর count এর সমান বা বড় হয় তাহলে ওই ইনডেক্সের ভ্যালু এক্সেস না করে লাইনটিকে বাইপাস করা হচ্ছে।

বিল্ট-ইন এক্সেপশন (Built-in Exceptions) ৷ঃ স্ট্যান্ডার্ড iOS ও OS X ফ্রেমওয়ার্ক এ অনেকগুলো বিল্ট-ইন এক্সেপশন রয়েছে যেগুলো দিয়ে মোটামুটি সবরকম এক্সেপশন @catch ব্লকে ধরা যায়। সবচেয়ে বেশী ব্যবহৃত এক্সেপশন গুলো সম্পর্কে নিচে ২-১ লাইন লেখা হলঃ

- NSRangeException ৷ঃ অ্যারে বা কালেকশনের বাইরের ইনডেক্সের ইলিমেন্ট এক্সেস করলে NSRangeException থ্রো হয়।
- NSInvalidArgumentException ৷ঃ মেথডে ইনভ্যালিড আরগুমেন্ট পাঠানো হলে NSInvalidArgumentException থ্রো হয়।
- NSInternalInconsistencyException ৷ঃ ইন্টারনালী যদি কোন অস্বাভাবিক কিছু ঘটে যা প্রোগ্রাম ক্র্যাশ করায় তখন NSInvalidArgumentException হয়।
- NSGenericException ৷ঃ যখন কোন স্পেসিফিক এক্সেপশন না ধরে জেনেরিক এক্সেপশন ধরতে হয় তখন NSGenericException ক্লাসের অবজেক্ট দিয়ে এক্সেপশনটি ধরা হয়। অর্থাৎ যেকোন কারনেই এক্সেপশন থ্রো হউক না কেন NSGenericException থ্রো হয়।

খেয়াল করুন যে এই সব এক্সেপশন গুলোর ভ্যালু স্ট্রিং। এদের NSException ক্লাসের সাবক্লাস ভাবার দরকার নেই। তাই কোন স্পেসিফিক এক্সেপশন পেতে চাইলে name প্রোপার্টি চেক করতে হবে।

```
// main.m
#import

int main(int argc, const char * argv[]) {
    @autoreleasepool {

        // menuList of the restaurant
        NSArray *menuList = @[@"Vegetable Rice", @"Chicken Toast", @"Thai Soup"];

        @try{
            // write codes where exception can be thrown
            NSLog(@"4th Item Name is %@", menuList[3]);
        }
        @catch(NSException *theException){
            // exception occurred. Lets handle this exception
            if(theException.name == NSRangeException)
                NSLog(@"NSRangeException Exception is caught.");
        }
        @finally{
            NSLog(@"Continues without crashing the application");
        }

        NSLog(@"This is being displayed. That means this application didnt crashed.");
    }
    return 0;
}
```

কাস্টম এক্সেপশন (Custom Exceptions) : `@throw` ডিরেক্টিভ দিয়ে সহজেই কাস্টম এক্সেপশন থ্রো করানো যায়। যেখানে এক্সেপশন থ্রো হবে সেখানে `NSException` ক্লাসের ইনস্ট্যান্স বানিয়ে `exceptionWithName:reason:userInfo:` ফ্যাক্টরী মেথড দিয়ে কাস্টম এক্সেপশন ইনস্ট্যান্স বানানো যায়। বোঝার জন্য নিচের উদাহরণটি দেখিঃ


```
// main.m
#import

NSString *getRandomItemFromMenuList(NSArray *menuList) {
    int max = (int)[menuList count];
    if (max == 0) {
        NSException *e = [NSException
                           exceptionWithName:@"EmptyMenuListException"
                           reason:@"*** The list has no food!"
                           userInfo:nil];

        @throw e;
    }
    int randomIndex = arc4random_uniform(max);
    return menuList[randomIndex];
}

int main(int argc, const char * argv[]) {
    @autoreleasepool {
        @try {
            NSString *foodItem = getRandomItemFromMenuList(@[]);
            NSLog(@"The selected food is: %@", foodItem);
        } @catch(NSException *theException) {
            if (theException.name == @"EmptyMenuListException") {
                NSLog(@"Caught an EmptyMenuListException");
            } else {
                NSLog(@"Ignored a %@ exception", theException);
                @throw;
            }
        }
    }
    return 0;
}
```

@throw ডিরেক্টিভ খুবই এক্সপেনসিভ অপারেশন। এই অপারেশনটি এক্সিকিউট করতে অপারেটিং সিস্টেম এর প্রচুর রিসোর্স খরচ হয়। তাই @throw ডিরেক্টিভ ব্যবহার না করে অবজেকটিভ সি এর ন্যাটিভ এক্সপেশন গুলো ব্যবহার করাই উত্তম।

এরর (**Errors**) :ঃ প্রোগ্রামে এমন অনেক অপারেশন থাকতে পারে যেগুলো ফেইল করলেও প্রোগ্রাম ক্র্যাশ করে না। এধরনের সমস্যাগুলো কে এরর বলা হয়। যেমন রানটাইমে ইউজারকে যদি কোন ফাইল ইনপুট দিতে বলা হয়, সেক্ষেত্রে যদি ফাইলটি করাণ্টেড থাকে বা ওই লোকেশনে ফাইলটি না পাওয়া যায় তাহলে এরর হয় কারন প্রোগ্রামটি ফাইলটিকে পড়তে পারে না। এরর হলে এক্সপেশনের মত প্রোগ্রাম ক্র্যাশ করে না। যখন কোন এরর হবে তখন ইউজারকে বোধগম্য মেসেজ এবং ইনস্ট্রাকশন দিয়ে প্রোগ্রামটি কন্টিনিও করানো কে এরর হ্যান্ডেলিং বলা হয়।

NSException ক্লাসের মতই NSError ক্লাসের ইনস্ট্যান্স দিয়ে ফেইল হওয়া অপারেশন তথা এরর সম্পর্কে যাবতীয় তথ্য পাওয়া যায়। NSError ক্লাসের প্রধান প্রোপার্টিগুলো নিচে দেওয়া হলঃ

- domain (NSString) :ঃ সব এরর গুলো কে কতগুলো ডোমেইনে ভাগ করা হয়েছে। এই ডোমেইন দিয়ে এরর গুলোকে সুন্দরভাবে hierarchy তে সাজানো হয়েছে।
- code (NSInteger) :ঃ code হল একটি নির্দিষ্ট ডোমেইনে এররের ইউনিক আইডি (ID)

- userInfo (NSDictionary) :ঃ NSError ক্লাসের মতই userInfo এর সম্পর্কে অতিরিক্ত তথ্য রাখে

আগেই বলেছি যে userInfo তে তথ্যগুলো কি-ভ্যালু পেয়ার (key-value pair) হিসেবে থাকে। আবার NSError এর তুলনায় NSError এ বেশী সংখ্যক তথ্য থাকে। গুরুত্বপূর্ণ কয়েকটি কি (key) হল NSLocalizedDescriptionKey, NSLocalizedFailureReasonErrorKey, NSUnderlyingErrorKey। মজার ব্যাপার হল এররের ধরন অনুযায়ী userInfo ডিকশনারীর কি(key) গুলো কমবেশী হয়। যেমন ফাইল লোড করতে গিয়ে যে এরর হয় তাতে NSFilePathErrorKey নামে একটি কি(key) থাকে যাতে ফাইলটির লোকেশন থাকে।

এরর হ্যান্ডেলিং (Handling Errors) :ঃ এরর হ্যান্ডেল করার জন্য এক্সপেশনের মত @try-@catch ব্লক বা আলাদা কিছু লিখতে হয় না। যেসব মেথড বা ফাংশনে এরর হতে পারে সেসব একটি বাদতি আর্গুমেন্ট হিসেবে NSError ক্লাসের অবজেক্ট পাঠানো হয়। মেথড অথবা ফাংশনটি যদি কোন কারনে ফেইল করে (কোন এরর হয়) তাহলে NO অথবা nil রিটার্ন করে এবং NSError ক্লাসের এই অবজেক্টটিতে এরর ডিটেইলস অ্যাসাইন হয়। কিন্তু যদি মেথড/ফাংশনটি সাকসেসফুলি এক্সিকিউট হয় তাহলে এই মেথডটি রিটার্ন টাইপ অনুযায়ী যে ডাটা রিটার্ন করার কথা ছিল তাই রিটার্ন করে।

অবজেক্টিভ সি এর বেশিরভাগ মেথডগুলো আর্গুমেন্ট হিসেবে NSError ক্লাসের অবজেক্টের Indirect রেফারেন্স নিয়ে থাকে। Indirect রেফারেন্স হল মূলত পয়েন্টার যা অন্য পয়েন্টারকে পয়েন্ট করে থাকে। এতে যে সুবিধা পাওয়া যায় তা হল, মেথডটি ওই আর্গুমেন্টটিকে NSError ক্লাসের সম্পূর্ণ নতুন একটি ইনস্ট্যান্সে পয়েন্ট করতে পারে। মেথডের মধ্যে ওই পয়েন্টার যে ইনস্ট্যান্সকে পয়েন্ট করে আছে তার প্রোপার্টিগুলোতে ভ্যালু অ্যাসাইন করা হয়। মেথডের এরর আর্গুমেন্টটি যদি Indirect রেফারেন্স এক্সপেক্ট করে তাহলে নিচের মত দুইবার পয়েন্টার চিহ্ন (double-pointer notation) দিয়ে ইনস্ট্যান্স পাওয়া যায়।

```
(NSError **)error
```

শুরুতেই এরর হ্যান্ডেল করার একটি সাদামাটা প্রোগ্রাম দেখা যাক। নিচের main.m ফাইলের মত করে আমাদের main.m ফাইলটি এডিট করি। এ প্রোগ্রামে শুরুতেই path নামের একটি ভ্যারিয়েবলে একটি ফাইলের পাথ দেওয়া হয়েছে। মূলত আমরা এই file.md ফাইলটির ডাটাগুলো পড়তে চাচ্ছি। এরপর NSError ক্লাসের একটি ইনস্ট্যান্স error নেওয়া হল। পরবর্তী লাইনে stringWithContentsOfFile:encoding:error: মেথডকে কিছু আর্গুমেন্ট সহ কল করা হয়েছে। এই মেথডটি অবজেক্টিভ সি এর বিস্টইন মেথড যা প্রথম আর্গুমেন্টের লোকেশন থেকে ফাইল লোড করে ফাইলের ডাটা (NSString টাইপ) রিটার্ন করে। তৃতীয় প্যারামিটারটি NSError ক্লাসের অবজেক্ট বা রেফারেন্স অপারেটর (&) দিয়ে পাঠানো হয়। &error মূলত একটি পয়েন্টার যা NSError ক্লাসের ইনস্ট্যান্স *error কে পয়েন্ট করে থাকবে।

```
// main.m
#import

int main(int argc, const char * argv[]) {

    @autoreleasepool {

        NSString *path = @"/path/to/file.md";

        NSError *error;
        NSString *content = [NSString stringWithContentsOfFile: path
                                                                encoding::NSUTF8StringEncoding
                                                                error: &error];

        if (content == nil) {
            // Method failed
            NSLog(@"Error loading file %@!", path);
            NSLog(@"Domain: %@", [error domain]);
            NSLog(@"Error Code: %ld", [error code]);
            NSLog(@"Description: %@", [error localizedDescription]);
            NSLog(@"Reason: %@", [error localizedFailureReason]);
            NSLog(@"Recovery procedure: %@", [error localizedRecoverySuggestion]);
        } else {
            // Method succeeded
            NSLog(@"Data Loaded from file: %@", content);
        }
    }
    return 0;
}
```

file.md ফাইলটি যদি ওই লোকেশনে না থাকে তাহলে stringWithContentsOfFile মেথডটি nil বা NO রিটার্ন করে এবং error অবজেক্টে এররের ডিটেইলস অ্যাসাইন হয়। অন্যথায় মেথডটি ফাইলের কনটেন্ট রিড করে তা রিটার্ন করে যা NSString টাইপ content ভ্যারিয়েবলে অ্যাসাইন হয়। এরপর content এর ভ্যালু চেক করে যদি NO বা nil হয় তাহলে error অবজেক্টের প্রোপার্টিগুলো code, domain, localizedDescription, localizedFailureReason ইত্যাদি এক্সেস করে এরর সম্পর্কে যাবতীয় তথ্য পাওয়া যায়।

যদি এমন হয় যে আপনি শুধু জানতে চাচ্ছেন যে এরর হয়েছে কিনা; এররের কারন না জানলেও চলবে সেক্ষেত্রে error ইনস্ট্যান্সের জন্য শুধুমাত্র NULL পাঠালেই চলবে। নিচের কোডটুকু খেয়াল করিঃ

```
NSString *content = [NSString stringWithContentsOfFile:fileToLoad
                                                                encoding:NSUTF8StringEncoding
                                                                error: NULL];

if (content == nil) {
    // error occurred
} else {
    // Method succeeded
}
```

বিল্ট-ইন এরর (**Built-in Errors**) : NSException ক্লাসের মত NSError ক্লাসের ও কিছু বিল্ট-ইন এরর ডোমেইন রয়েছে যা দিয়ে এরর গুলো কে রিপ্রেজেন্ট করা যায়। প্রধান কয়েকটি বিল্ট-ইন এরর ডোমেইন এর নাম নিচে দেওয়া হলঃ

- NSMachErrorDomain
- NSPOSIXErrorDomain
- NSOSStatusErrorDomain
- NSCocoaErrorDomain

আমাদের প্রোগ্রামগুলোতে সবচেয়ে বেশী যে এরর গুলো ঘটবে সেগুলো NSCocoaErrorDomain এর অন্তর্ভুক্ত। তারপরও চাইলে আরো লো-লেভেলের এরর ডোমেইনের এরর ইনফরমেশন পাওয়া যাবে। যেমন আমাদের উপরের main.m ফাইলে যদি নিচের লাইনটি লেখি তাহলে NSPOSIXErrorDomain এর এরর ইনফরমেশন পাওয়া যাবে।

```
NSLog(@"Lower Level Domain Error : %@",error.userInfo[NSUnderlyingErrorKey]);
```

এই লাইনটির আউটপুট হবে **Lower Level Domain Error : Error**

Domain=NSPOSIXErrorDomain Code=2 "The operation couldn't be completed. No such file or directory"

আগেই বলেছি যে সকল এরর গুলো কতগুলো ডোমেইন এ ভাগ করা হয়েছে। প্রত্যেক ডোমেইনের অন্তর্গত সকল এররের নির্দিষ্ট ও ইউনিক কোড (code) থাকে যা দিয়ে স্পেসিফিক এরর ইনফরমেশন পাওয়া যায়। [Foundation Constants Reference](#) ডকুমেন্টটিতে কিছু enum দিয়ে NSCocoaErrorDomain ডোমেইনের প্রায় সকল কোডের (code) বর্ণনা দেওয়া আছে। উদাহরণ স্বরূপ নিচের প্রোগ্রামে দেখা যাচ্ছে NSCocoaErrorDomain এর NSFileReadNoSuchFileError নামক কোডের এরর খোজার জন্য প্রোগ্রামটি ওত পেতে বসে আছে।

```
if (content == nil) {
    if ([error.domain isEqualToString:@"NSCocoaErrorDomain"] &&
        error.code == NSFileReadNoSuchFileError) {
        NSLog(@"That file doesn't exist!");
        NSLog(@"Path: %@", [[error userInfo] objectForKey:NSFilePathErrorKey]);
    } else {
        NSLog(@"Some other kind of read occurred");
    }
}
```

কাস্টম এরর (**Custom Errors**) : বড় সাইজের iOS ও OS X এর অ্যাপ্লিকেশনে বিভিন্ন মেথডে বিভিন্ন ধরনের এরর হওয়ার সম্ভাবনা থাকে যা সংখ্যায় খুব কম নয়। একারণে প্রোগ্রামারের সুবিধার্থে ও ইউজারের কাছে আরও সহজবোধ্য মেসেজ পাঠানোর উদ্দেশ্যে কাস্টম এরর বানানো হয়। বেস্ট প্র্যাকটিস হিসেবে সব এররগুলো কে একটি আলাদা হেডার ফাইলে (For Ex- RestaurantErrors.h) লেখা হয়। পরবর্তীতে প্রোগ্রামের যেসব ইমপ্লিমেন্টেশন ফাইলে এই কাস্টম এরর গুলো ব্যবহার করা হবে সেখানে ইমপোর্ট করে নিলেই হবে।

```
//RestaurantErrors.h
NSString *RestaurantErrorDomain = @"net.Nuhil.Restaurant.ErrorDomain";

enum {
    MenuListNotLoadedError,
    MenuListEmptyError,
    RestaurantInternalError
};
```

কাস্টম এরর ডোমেইনের নাম যেকোন কিছুই হতে পারে, তবে

CompanyName.ProjectOrFrameWorkName.ErrorDomain রাখাটাই বেস্ট প্র্যাকটিস। এরর কোড কনস্ট্যান্ট গুলো কে enum এর মধ্যে ডিফাইন করা হয়।

```

// main.m
#import
#import "RestaurantErrors.h"

NSString *getFoodTitleWithId( int foodId, NSError **error) {

    NSArray *Food = @[];

    int max = [Food count];

    if (max == 0) {
        if (error != NULL) {
            NSDictionary *userInfo = @{NSLocalizedStringKey: @"MenuList"
                                       " Currently Empty."};

            *error = [NSError errorWithDomain: RestaurantErrorDomain
                                       code: MenuListEmptyError
                                       userInfo: userInfo];
        }
        return nil;
    }
    return Food[foodId];
}

int main(int argc, const char * argv[]) {
    @autoreleasepool {
        int searchFor = 2;
        NSError *error;
        NSString *title = getFoodTitleWithId(searchFor, &error);

        if (title == nil) {
            // If Failed
            NSLog(@"Title could not be found");
            NSLog(@"Domain: %@", error.domain);
            NSLog(@"Error Code: %ld", error.code);
            NSLog(@"Description: %@", [error localizedDescription]);

        } else {
            // Succeeded
            NSLog(@"Selected Food Title Found!");
            NSLog(@"%@", title);
        }
    }
    return 0;
}

```

main.m এর main মেথড এ দেখা যাচ্ছে যে আরগুমেন্ট হিসেবে একটি ইনডেক্স বা আইডি (searchFor) এবং NSError ক্লাসের একটি Indirect রেফারেন্স &error সহ getFoodTitleWithId কে কল করা হয়েছে। মেথডটিতে &error পাঠানোর উদ্দেশ্য হল যদি মেথডের ভেতর কোন এরর হয় তাহলে এররের প্রোপার্টিগুলোর ভ্যালু যাতে এই

error ইনস্ট্যান্স এ অ্যাসাইন হয়। যদি ২য় আরগুমেন্টে NULL পাঠানো হত তাহলে এরর সম্পর্কিত কোন ইনফরমেশন আর পাওয়া যেত না। যদি এরর হত তাহলে মেথডটি শুধুমাত্র NO অথবা nil রিটার্ন করত।

getFoodTitleWithId মেথডে FoodId তে 2 এবং NSArray ক্লাসের Indirect রেফারেন্স প্যারামিটার হিসেবে এসেছে। মেথডের শুরুতেই NSArray টাইপ একটি ভ্যারিয়েবল ডিক্লেয়ার করা হয়েছে যাতে আইটেম সংখ্যা শূন্য। অর্থাৎ পরের লাইনে max এর ভ্যালু শূন্য হবে। এপর্যায়ে আমরা চাচ্ছি যেহেতু Food লিস্টটিতে কোন আইটেম নেই তাই একটি কাস্টম এরর থ্রো করার। তাই যখন max এর ভ্যালু শূন্য এবং NSError এর Indirect রেফারেন্সটি NULL নাহয় তখন কাস্টম এররটির প্রোপার্টিগুলোর ভ্যালু পপুলেট করা হচ্ছে এবং অবশেষে nil রিটার্ন করে দেওয়া হয়েছে।

উপরের প্রোগ্রামটি রান করলে নিচের মত আউটপুট আসবে।

```
2014-06-10 00:56:19.817 restaurant[904:303] Title could not be found
2014-06-10 00:56:19.819 restaurant[904:303] Domain: net.Nuhil.Restaurant.ErrorDomain
2014-06-10 00:56:19.819 restaurant[904:303] Error Code: 1
2014-06-10 00:56:19.820 restaurant[904:303] Description: MenuList Currently Empty.
Program ended with exit code: 0
```

RestaurantErrorDomain এ এরর কোডগুলো যেহেতু enum টাইপ সেহেতু এরর কোডগুলোর ভ্যালুগুলো শূন্য থেকে শুরু করে ২ পর্যন্ত। তাই MenuListEmptyError এর ভ্যালু ১।

পরিশিষ্টঃ এই চ্যাপ্টারের মাধ্যমেই আমাদের ব্যাসিক ওজ্জেক্টিভ-সি লার্নিং সেকশন শেষ হয়ে যাচ্ছে। এর পরের সেকশনে আসছে রিয়াল লাইফ গ্রাফিক্যাল অ্যাপ্লিকেশন তৈরি সম্পর্কিত ১০টি চ্যাপ্টার। উক্ত সেকশন শেষ হবার পর আসবে ব্যাসিক সুইফট ল্যান্ডুয়েজ লার্নিং সেকশন।

অনেকেই হয়ত ভেবে থাকতে পারেন যে, Apple যেহেতু iOS অ্যাপ ডেভেলপমেন্ট এর জন্য তাদের নতুন ল্যান্ডুয়েজ ঘোষণা করেছে তাহলে আর অবজেক্টিভ-সি শিখে কি লাভ। কিন্তু এ ধারণা একদম ভুল। প্রথমত, এখন পর্যন্ত অ্যাপ স্টোরে ১০ লাখেরও বেশি অ্যাপ জীবন্ত আছে যেগুলো অবজেক্টিভ-সি তে করা। সেই অ্যাপ গুলোর মেইন্টেনেন্স, আপডেট, বাগ ফিক্সিং ইত্যাদির জন্য আরও সামনে ৬/৭ বছর iOS এবং অবজেক্টিভ-সি এর ব্যবহার চলবেই। দ্বিতীয়ত, তাদের নতুন ঘোষণা মতে, একটি প্রজেক্টে অথবা অ্যাপ্লিকেশনে একি সাথে অবজেক্টিভ-সি এবং সুইফট ল্যান্ডুয়েজ কোড থাকতে পারে এবং রান হতে পারে। অর্থাৎ তারা চাচ্ছে, আস্তে আস্তে অবজেক্টিভ-সি এর জায়গাটা কমে আসুক, একবারে না। তৃতীয়ত, অনেক ডেভেলপার বা কোম্পানি হয়ত চাইতেই পারে যে তাদের সামনের অ্যাপ্লিকেশনগুলো তারা অবজেক্টিভ-সি দিয়েই করবে কারন সুইফট -এ দক্ষ ডেভেলপার এই মুহূর্তে হয়ত তারা ব্যবস্থা করতে পারবেন না। চতুর্থত, হাজার হাজার থার্ড পার্টি লাইব্রেরী, ফ্রেমওয়ার্কও কিন্তু অবজেক্টিভ-সি তে করা যেগুলো হয়ত আপনি আপনার নেক্সট প্রজেক্টে ব্যবহার করবেন বলে ভেবে রেখেছেন। অন্যদিকে, অনেকেই হয়ত চাইবেন তাদের পরবর্তী প্রজেক্টটি নতুন ল্যান্ডুয়েজ দিয়ে করতে। অর্থাৎ আপনি যদি ফ্রেশ ভাবে iOS অ্যাপ ডেভেলপমেন্ট শুরু করতে চান তাহলে শুধুমাত্র সুইফট শেখা শুরু করতে পারেন কিন্তু যদি ক্যারিয়ার হিসেবে দেখতে চান তাহলে আপনাকে দুটো ল্যান্ডুয়েজ সম্পর্কে ধারণা রাখতেই হবে।

Originally Posted [Here](#)

এই সেকশনে থাকছে

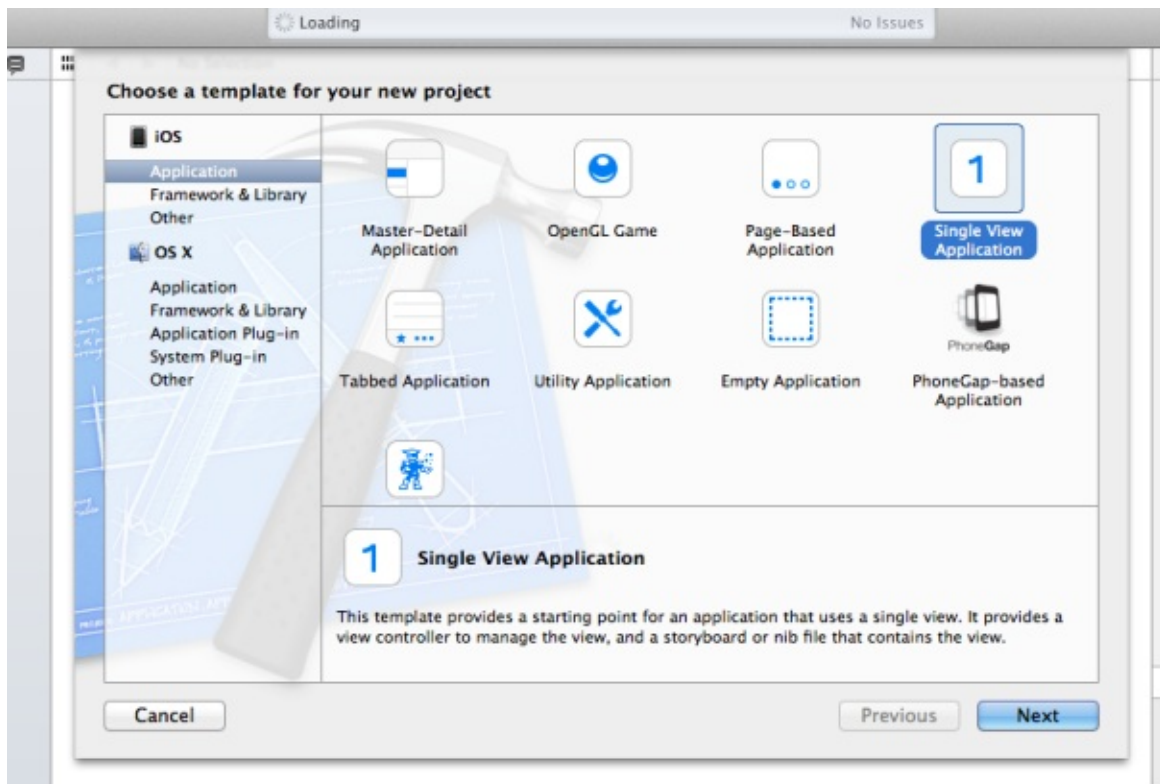
- টুলস সেটআপ এবং একটি সাধারণ হ্যালো ওয়ার্ল্ড অ্যাপ তৈরি
- iOS অ্যাপে ব্যাসিক ইনপুট আউটপুট ও কিবোর্ড হ্যান্ডেলিং
- আরও চ্যাপ্টার আসছে ...

অবেজেক্টিভ-সি দিয়ে iOS7 এর জন্য প্রথম হ্যালো ওয়ার্ল্ড এপ্লিকেশন যেভাবে করব

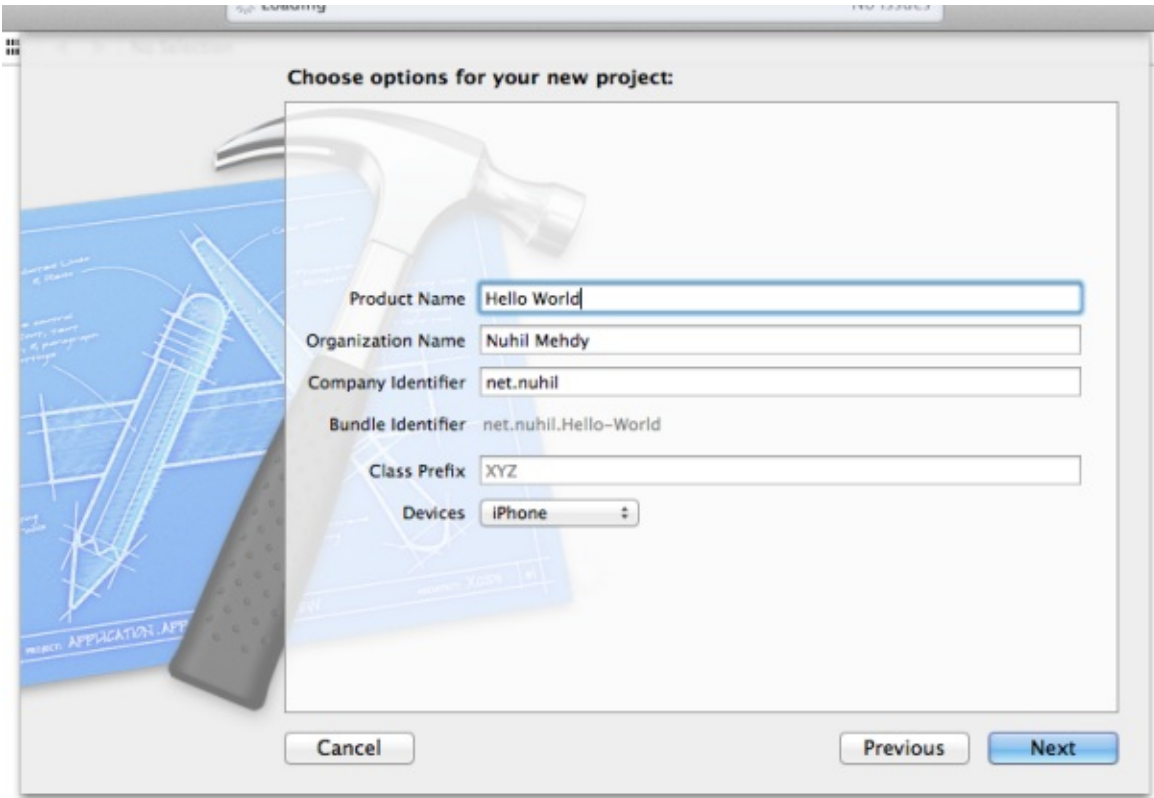
ভূমিকাঃ স্বাগত জানাচ্ছি আপনার iOS এপ্লিকেশন নিয়ে আগ্রহের জন্য। এটি আমাদের উপরে উল্লেখিত সিরিজের দ্বিতীয় সেকশনের প্রথম চ্যাপ্টার। প্রথম সেকশনের ১০টি চ্যাপ্টারের মাধ্যমে সহজ বাংলায় অবজেক্টিভ-সি এর ব্যাসিক লার্নিং কোর্স সম্পন্ন হয়েছে। আপনি যদি ওই সেকশনের কোর্স গুলো ফলো করে না থাকেন তাহলে উপরের সূচি ঘুরে আসতে পারেন। আজকে আমরা শেয়ার করব কিভাবে আপনি Xcode 5 এ অবজেক্টিভ-সি দিয়ে iOS7 এর জন্য সহজ একটি হ্যালো ওয়ার্ল্ড এপ্লিকেশন তৈরি করতে পারেন।

কি কি জিনিস অবশ্যই লাগবেঃ ১. **Mac OS X** চালিত ম্যাক কম্পিউটার সিস্টেম - এটাই প্রথম ও প্রধান দরকারী জিনিস। আপনি একটি ম্যাক বুক, ম্যাক বুক প্রো, আই ম্যাক অথবা ম্যাক মিনি ব্যবহার করতে পারেন এবং সেটাতে সর্বশেষ হালনাগাদ করা (OS X 10.8.5 Mountain Lion) অপারেটিং সিস্টেম ব্যবহার করতে পারেন। ২. একটি ফ্রি **Apple** ডেভেলপার একাউন্ট - নতুন একাউন্ট খুলতে পারেন [এখান থেকে](#)। ৩. **Xcode 5** - আমি আগেই বলেছি এখানে আলোচিত সব গুলো পোস্ট লেখা হবে Xcode 5 এ কোড করে ও এর অন্যান্য সুবিধা গুলো ব্যবহার করে। আপনার যদি ম্যাক থাকে এবং সেটার অপারেটিং সিস্টেম 10.8.4 এর পরের যেকোনো ভার্সন হয় তাহলে আপনি সেটাতে Xcode 5 ইনস্টল/আপডেট করতে পারবেন। Xcode প্রথমত একটি ইন্টিগ্রেটেড ডেভেলপমেন্ট এনভায়রনমেন্ট বা আই, ডি, ই। এটার সাথে iOS এবং OSX এর জন্য সফটওয়্যার ডেভেলপমেন্ট কিট বা এস, ডি, কে টিও থাকে। উপরন্তু এটি একটি একটি সোর্স কোড এডিটর যার সাথে ডিবাগিং টুল এবং সুন্দর একটি ইউজার ইন্টারফেস (ইউ, আই) বিল্ডার এবং একটি এপ্লিকেশন সিমুলেটরও আছে। আপাতত আর কিছু না।

শুরু করি হ্যালো ওয়ার্ল্ডঃ Xcode চালু করুন এবং File মেনু থেকে New অপশন এর Project মেনু সিলেক্ট করুন। নিচের মত একটি স্ক্রিন আসবে।

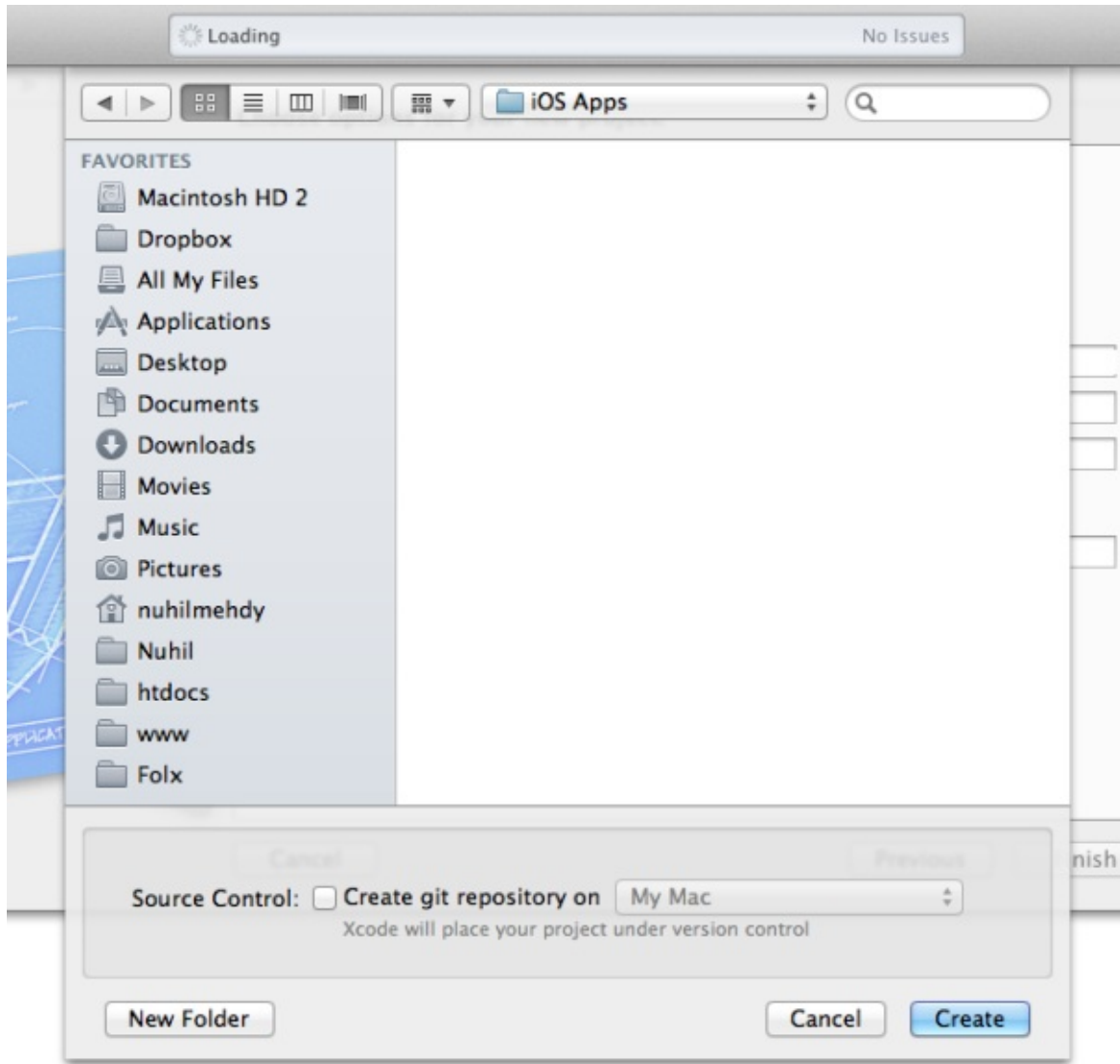


খেয়াল করুন বাম পাশে যাতে iOS এর নিচে Application সিলেক্ট থাকে এবং ডান পাশ থেকে Single View Application সিলেক্ট করে Next বাটনে ক্লিক করুন। এবার নিচের মত আরেকটি স্ক্রিন আসবে।



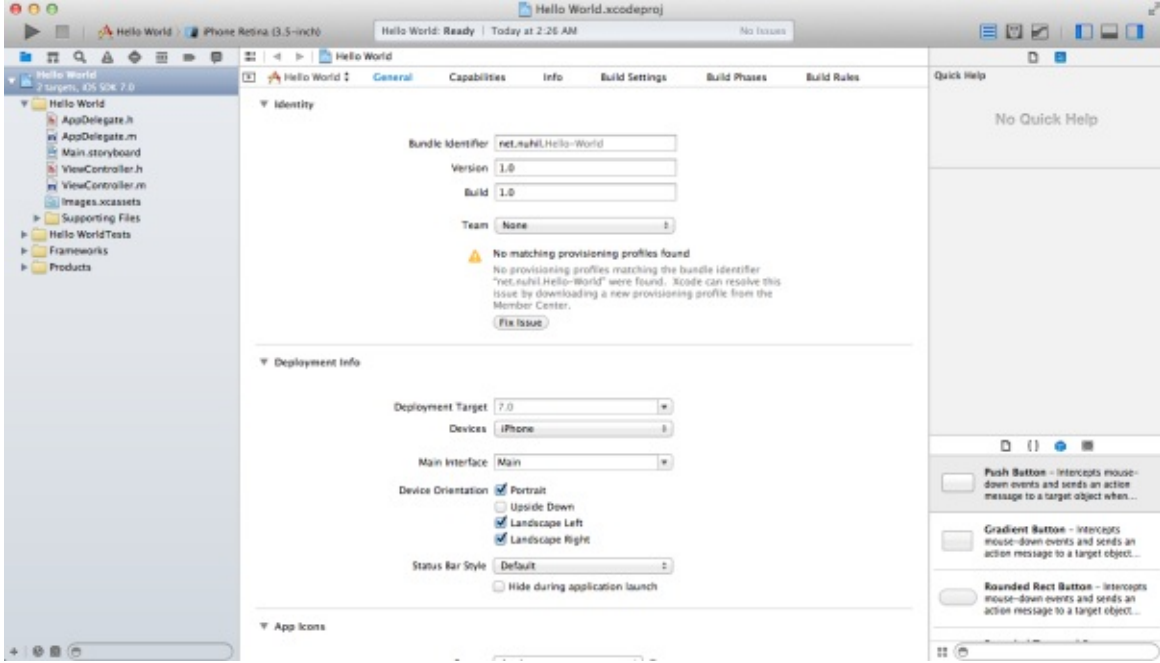
এটার অপশনগুলো নিজের মত করে পূরণ করতে পারেন। Product Name দিন খুসি মত যেটা আসলে এপ্লিকেশন এর নাম। Organization Name ও দিন আপনার পছন্দ মত। Company Identifier ফিল্ডে মূলত একটি ডোমেইন উল্টো ভাবে লিখতে হয়। আপনার যেকোনো একটি ডোমেইন থাকলে সেটা এই ভাবে লিখতে পারেন অথবা না থাকলে এই ফিল্ডে edu.self লিখতে পারেন। Class Prefix ঘর খালি রাখতে পারেন অথবা যেকোনো কিছু লিখতে পারেন যেটা এই এপ্লিকেশন এর জন্য প্রয়োজনীয় ক্লাস গুলোর নামের আগে সয়ংক্রিয় ভাবে যুক্ত হয়ে যাবে।

আমি খালি রাখছি। Devices হিসেবে সিলেক্ট করুন iPhone। এরপর Next বাটনে ক্লিক করুন। এখন নিচের মত যে স্ক্রিন আসবে সেখানে;



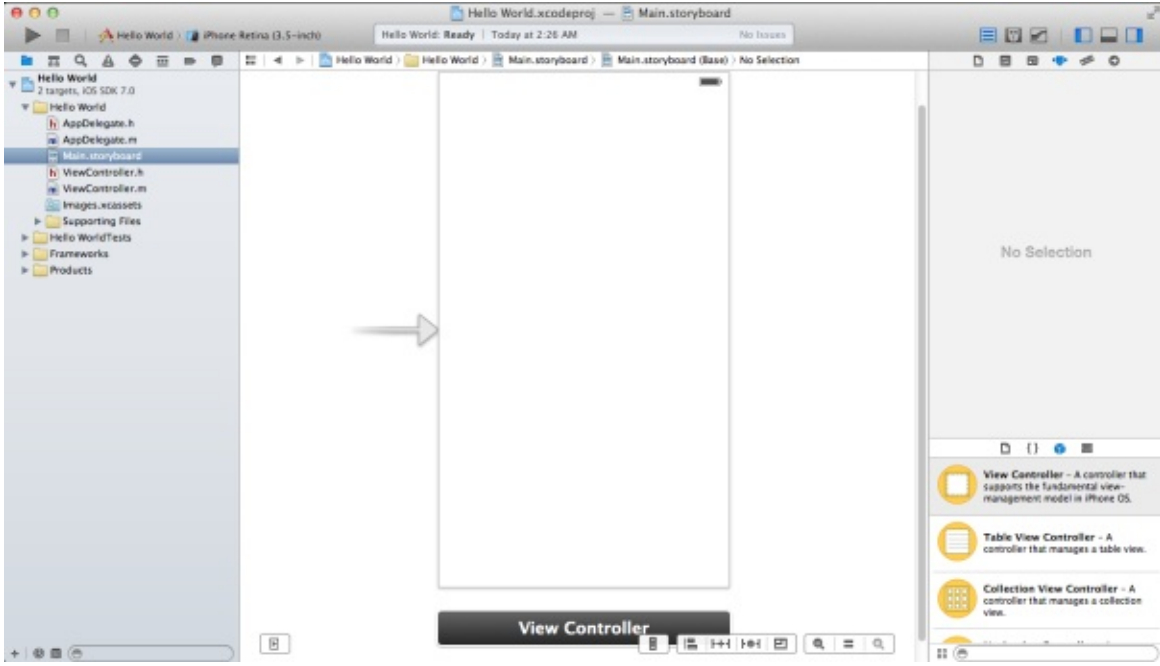
আপনার প্রজেক্টটি কোথায় সেভ করতে চান সেই লোকেশন দেখিয়ে দিয়ে Create বাটনে ক্লিক করুন। এখানে আরেকটি সুন্দর অপশন থাকে যেটা হচ্ছে Create git repository on My Mac যেটি ইচ্ছা করলে সিলেক্ট করতেও পারেন যদি পুরো প্রজেক্ট ডিরেক্টরিটিকে git ভার্সন কন্ট্রোলিং দিয়ে পরবর্তিতে ম্যানেজ করতে চান। আপাতত আমি

সেটা করছি না। Create বাটনে ক্লিক করার পরই নিচের মত স্ক্রিন তৈরী হয়ে যাবে যেখানে আপনি আপনার এপ্লিকেশনের কোডিং, ইউ, আই বিল্ডিং, ডিবাগিং সহ অনেক কিছুই করতে পারবেন।

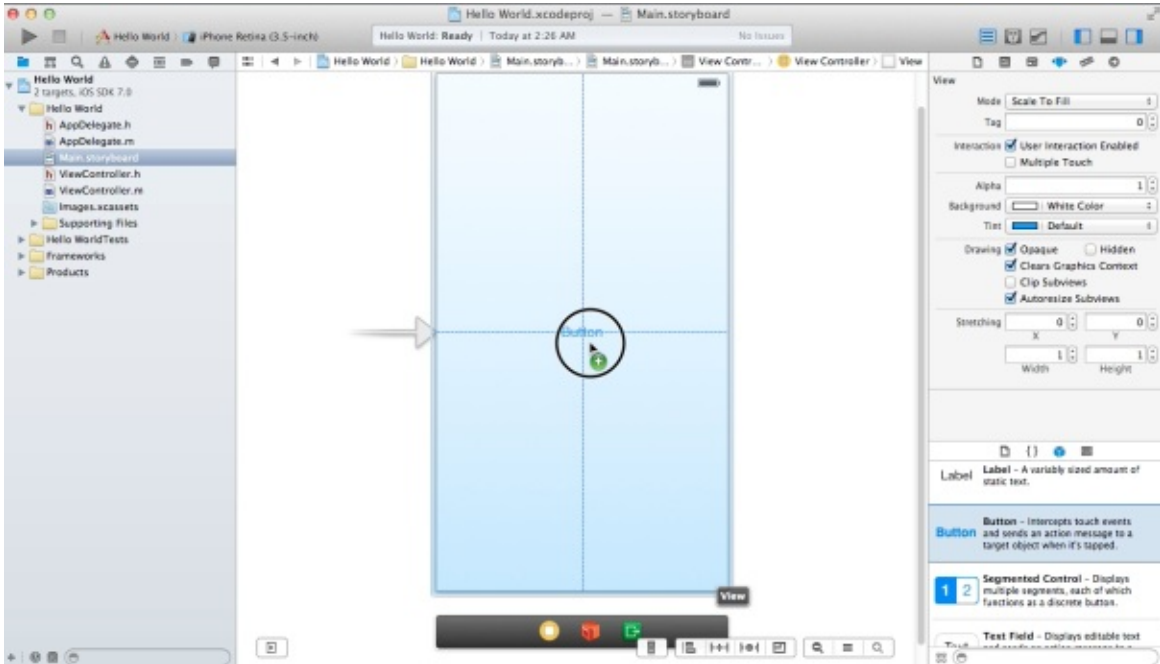


তিন কলাম বিশিষ্ট এই উইনডো এর বাম পাশে আপনি এই প্রজেক্ট এর জন্য প্রয়োজনীয় ক্লাস ফাইল, ফ্রেমওয়ার্ক, সাপোর্টিং ফাইল ইত্যাদি দেখতে পারবেন। এটাকে প্রজেক্ট ন্যাভিগেটর বলে। মাঝের অংশটিকে বলা হয় এডিটিং এরিয়া যেখানে একদিকে যেমন কোড লিখতে পারবেন তেমনি প্রজেক্ট এর বিভিন্ন সেটিংস পরিবর্তন করার সময়ও এখানে সেই প্যানেলটি চলে আসবে। আর ডানের অংশকে বলা হয় ইউটিলিটি এরিয়া যেখান থেকে আপনি কোনো ইউজার ইন্টারফেস ফাইল এর ক্লাস এসাইন করে দিতে পারেন এবং ইউজার ইন্টারফেস এর বিভিন্ন কিট যেমন বাটন, লেবেল, নতুন একটি ডিউ এসব মাঝখানের এডিটর-এ নিয়ে এসে কাজ করতে পারেন অথবা এগুলোর সাইজ ও অন্যান্য প্রপার্টি ঠিক ঠাক করতে পারেন। এই তিন কলাম এর উপর দিয়ে যে একটি বো সেটাকে টুলবার বলা হয় যেখানকার টুল গুলোর ব্যবহার আপনি নিজেই একটু খেয়াল করে ঘেটে দেখলেই বুঝে যাবেন। হাজার হলেও আপনি একজন iOS ডেভেলপার :P

ইউজার ইন্টারফেস তৈরিঃ আসুন আমাদের ছোটো এপ্লিকেশনটির জন্য গ্রাফিকাল ইউজার ইন্টারফেস ডিজাইন করি। প্রজেক্ট ন্যাভিগেটর থেকে Main.storyboard ফাইলটি সিলেক্ট করুন। সাথে সাথে ডান পাশে একটি সাদা এপ্লিকেশন স্ক্রিন চলে আসবে।

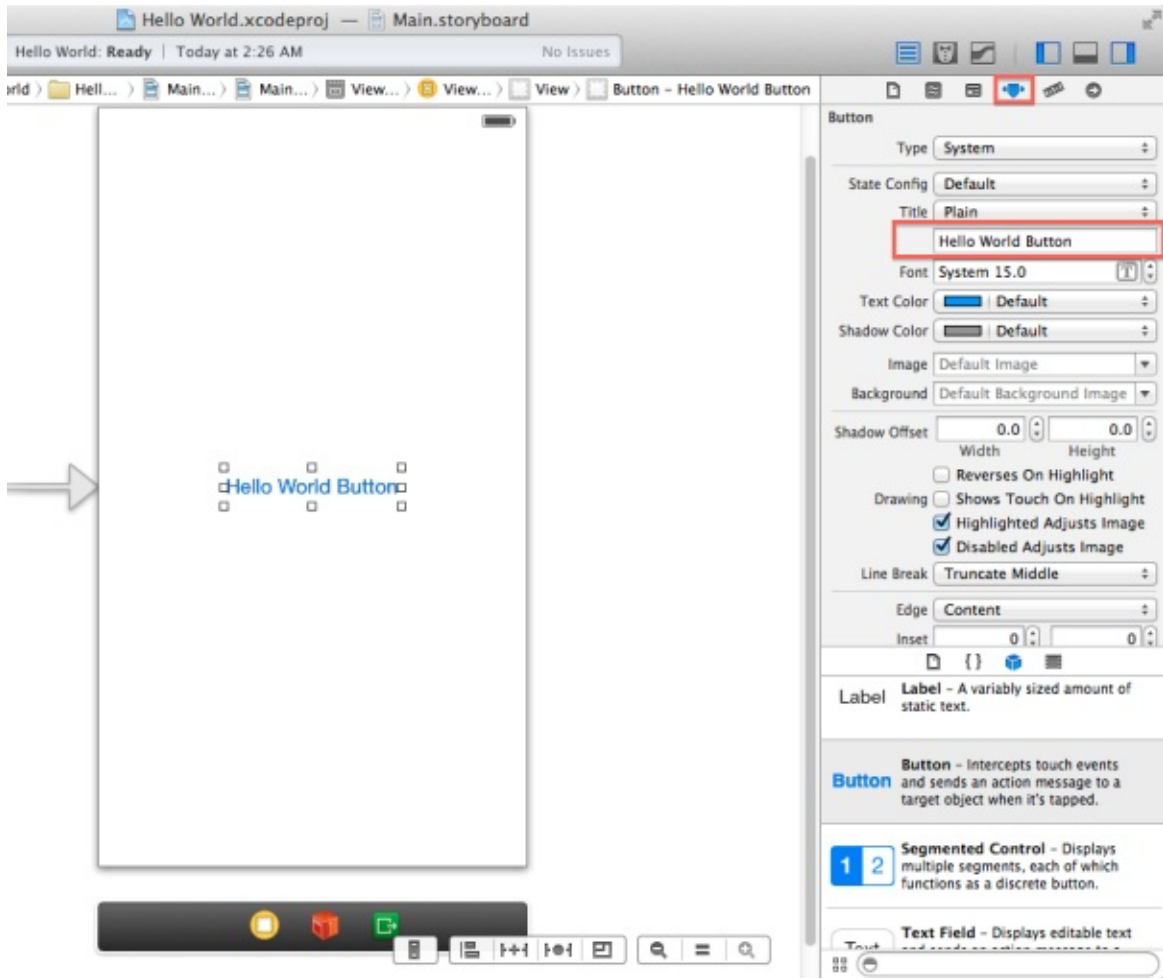


Main.storyboard মূলত একটি ইউজার ইন্টারফেস ফাইল যা কিনা Xcode আমাদেরকে গ্রাফিক্যালি দেখাচ্ছে এবং এই স্টোরিবোর্ড-এ আমরা ইচ্ছা মত নতুন নতুন এপ্লিকেশন স্ক্রিন নিয়ে সেই স্ক্রিন বা সিন (Scene) গুলোর মধ্যে বিভিন্ন ধরনের রিলেশন বা কানেকশন তৈরী করতে পারি এবং প্রত্যেকটি স্ক্রিন এর মধ্যে অন্যান্য ইউ, আই কিট এলিমেন্ট যেমন বাটন, লেবেল, টেক্সট ফিল্ড ইত্যাদি যুক্তও করতে পারি। যাই হোক আপাতত ধরে নিন আমাদের এপ্লিকেশন এর একটাই স্ক্রিন/সিন যেটা আমরা ইতোমধ্যে দেখতে পাচ্ছি। এবার ডান পাশের ইউটিলিটি এরিয়ার নিচের দিক থেকে একটি বাটন এলিমেন্ট টেনে ধরে এনে আমাদের একমাত্র স্ক্রিন এর উপর ছেড়ে দিন। নিচের ছবির মত।

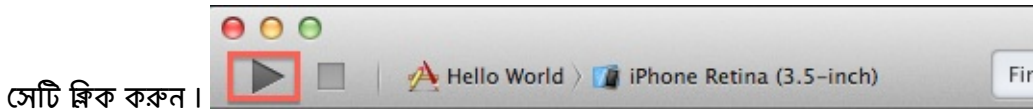


খুব খেয়াল রাখবেন যেন স্ক্রিন/সিন-টি পর্যাপ্ত জুম ইন অবস্থায় থাকে অর্থাৎ নিচের ছবির মত। স্ক্রিনটি যদি স্টোরি বোর্ডে জুম আউট অবস্থায় থাকে তাহলে আপনি বাটন এলিমেন্ট টেনে এনে স্ক্রিন/সিন এর উপর ছেড়ে দিলেও সেটা সেখানে বসতে চাইবে না। বাটন এলিমেন্টটি সিলেক্ট থাকা অবস্থায় ডান পাশের এটিবিউট ইমপেক্টর এবং সাইজ

ইন্সপেক্টর ব্যবহার করে সেটার সাইজ, টাইটেল, টেক্সট ইত্যাদি পরিবর্তন করতে পারবেন। আপাতত বাটনটির টাইটেল দিন Hello World Button নিচের ছবির মত।

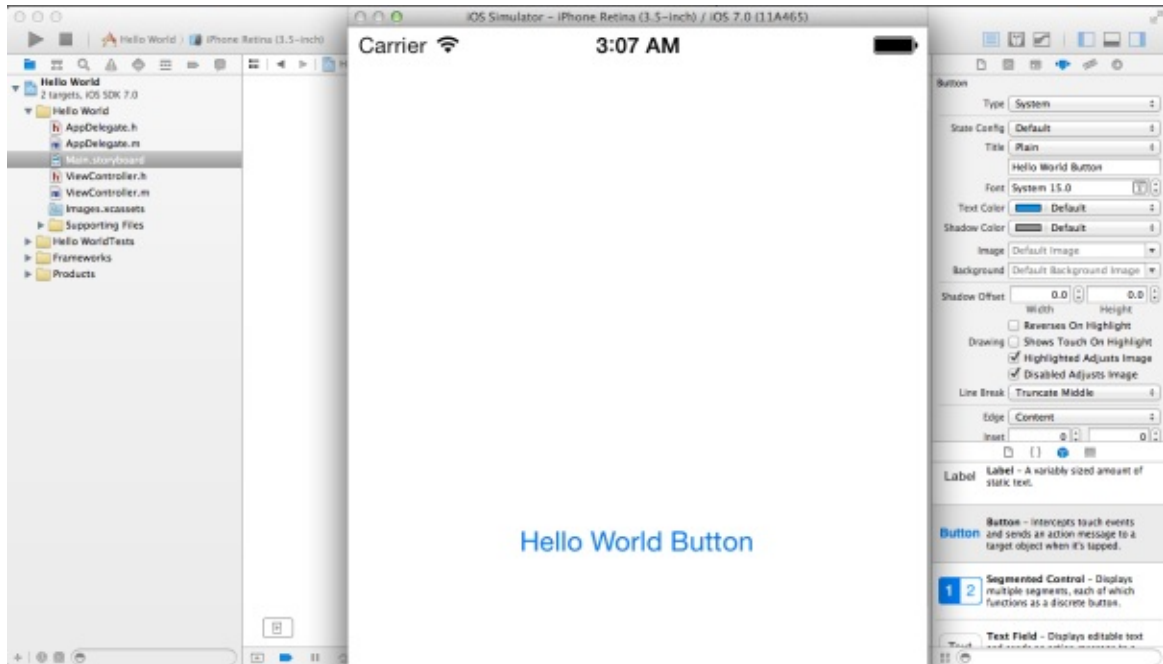


পরীক্ষামূলক ও প্রথম রানঃ এই অবস্থায় Xcode এর বাম পাশে উপরের দিকে প্লে বাটন এর মত একটি বাটন আছে



সেটি ক্লিক করুন।

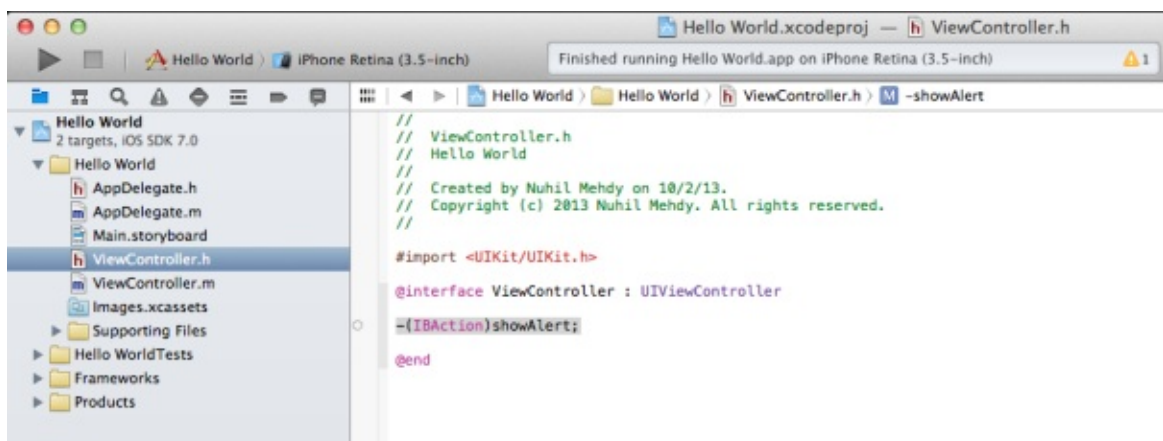
এটা একইসাথে এই প্রজেক্টকে বিল্ড (কম্পাইল ও প্যাকেজিং) করবে এবং রান করবে এবং সব কিছু ঠিক থাক থাকলে Xcode এই এপ্লিকেশন কে iOS বা iPhone সিমুলেটর-এ চালিয়ে দেখাবে নিচের ছবির মত।



এবার এর টুলবারে প্লে বাটনের পাশে থাকা স্টপ বাটনে ক্লিক করে সিমুলেটরটি এবং রানিং এপ্লিকেশন বন্ধ করুন

অল্প কিছু কোডিংঃ এতক্ষণ পর্যন্ত আমরা এক লাইনও objective-c ল্যাংগুয়েজ দিয়ে কোড লিখি নাই। তারপরেও একটা হ্যালো ওয়ার্ল্ড এপ্লিকেশন তৈরী করা গেছে। এবার আমরা অল্প কিছু কোড লিখব যাতে করে আমরা Hello World Button বাটনটিকে ট্যাপ (সিমুলেটর এর হিসেবে ক্লিক) করলে iPhone এর ডিফল্ট একটি এলার্ট ম্যাসেজ দেখতে পাই। এখন প্রজেক্ট ন্যাভিগেটর থেকে ViewController.h ফাইলটি সিলেক্ট করুন। মাঝখানের এডিটর এরিয়াতে এটির মধ্যে থাকা টেম্পলেট (সবনিম্ন প্রয়োজনীয়) কোড টুকু দেখতে পারবেন। এটির @end এর আগে নিচের কোডটি লিখুন এবং Command+s চেপে ফাইলটি সেভ করুন,

```
objective-c -(IBAction)showAlert;
```



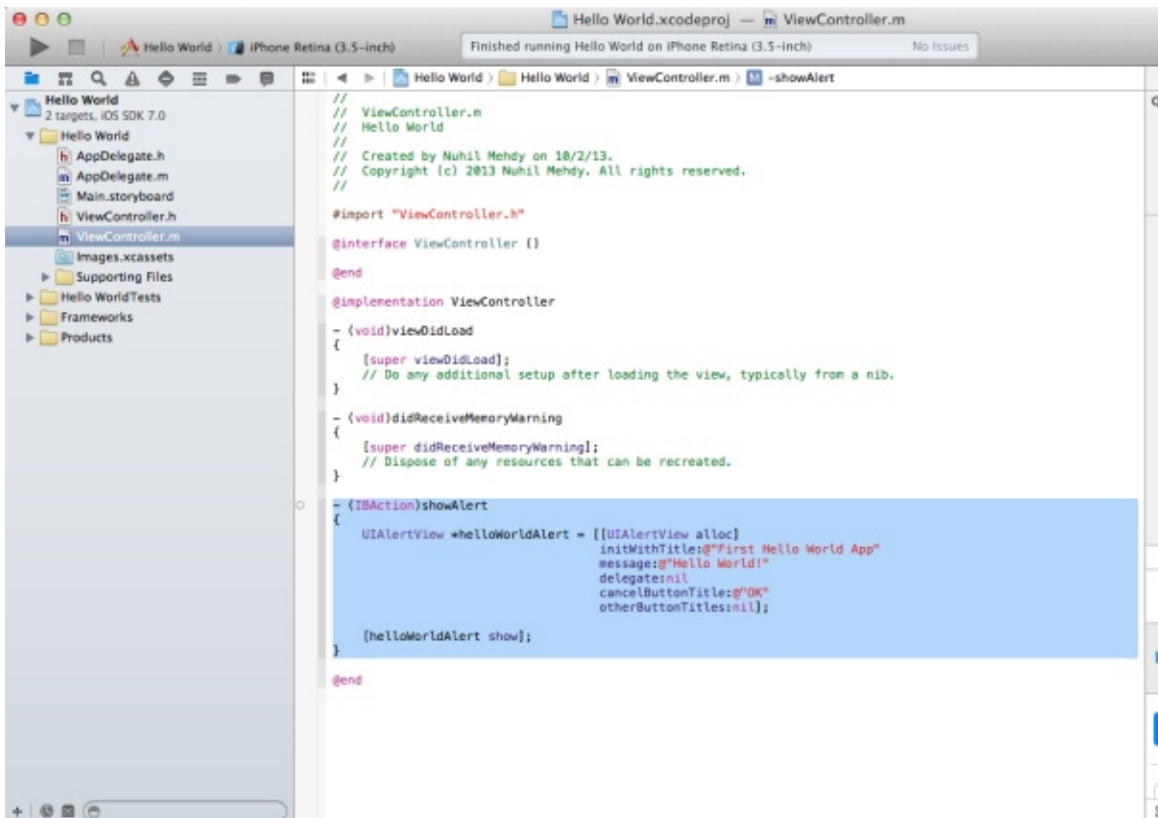
এবার প্রজেক্ট ন্যাভিগেটর থেকে ViewController.m ফাইলটি সিলেক্ট করুন। মাঝখানের এডিটর এরিয়াতে এটির মধ্যে থাকা টেম্পলেট (সবনিম্ন প্রয়োজনীয়) কোড টুকু দেখতে পারবেন। এটির @implementation এর @end এর আগে নিচের কোডটি লিখুন এবং Command+s চেপে ফাইলটি সেভ করুন,


```

- (IBAction)showAlert
{
    UIAlertView *helloWorldAlert = [[UIAlertView alloc]
                                     initWithTitle:@"First Hello World App"
                                     message:@"Hello World!"
                                     delegate:nil
                                     cancelButtonTitle:@"OK"
                                     otherButtonTitles:nil];

    [helloWorldAlert show];
}

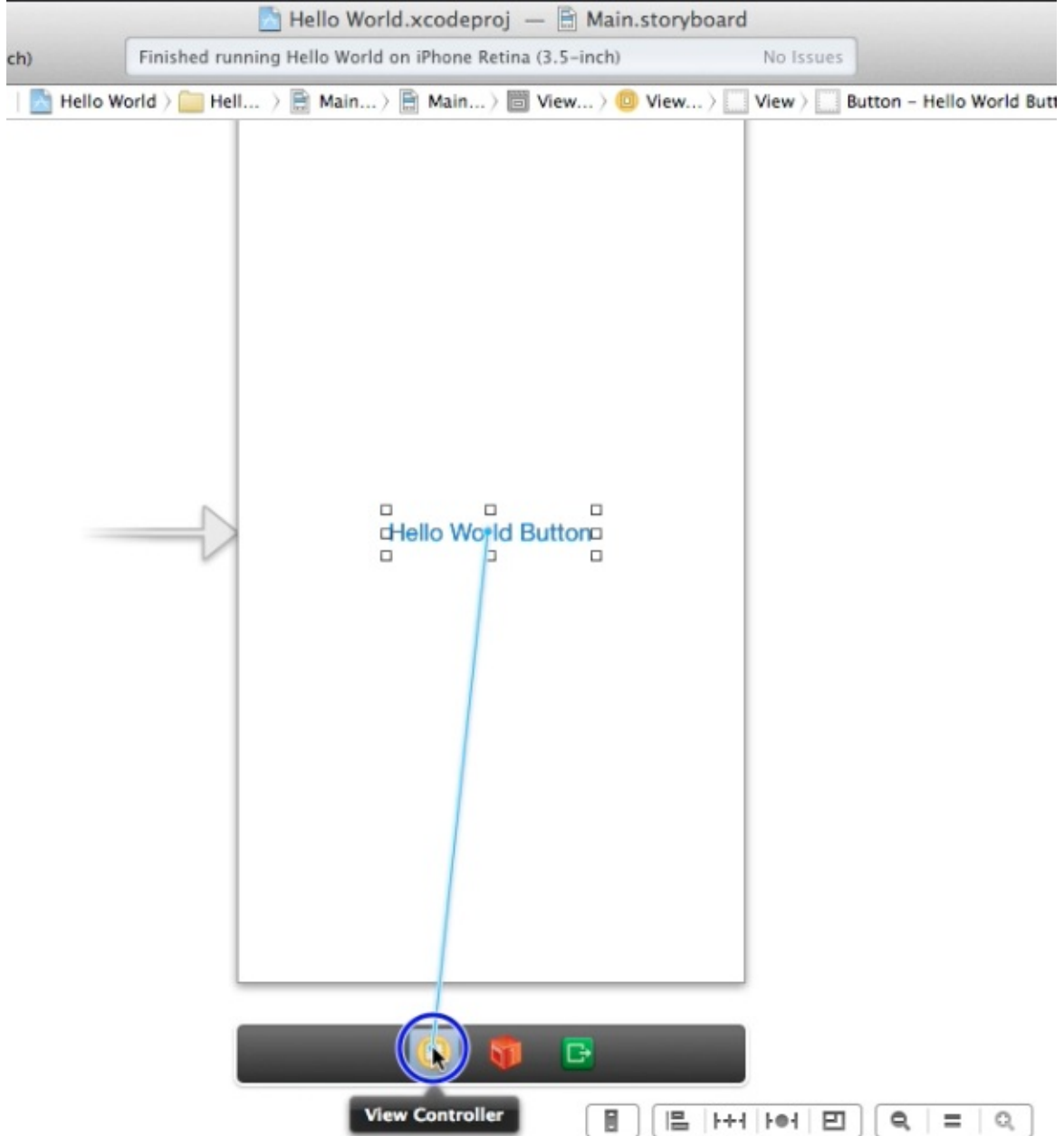
```



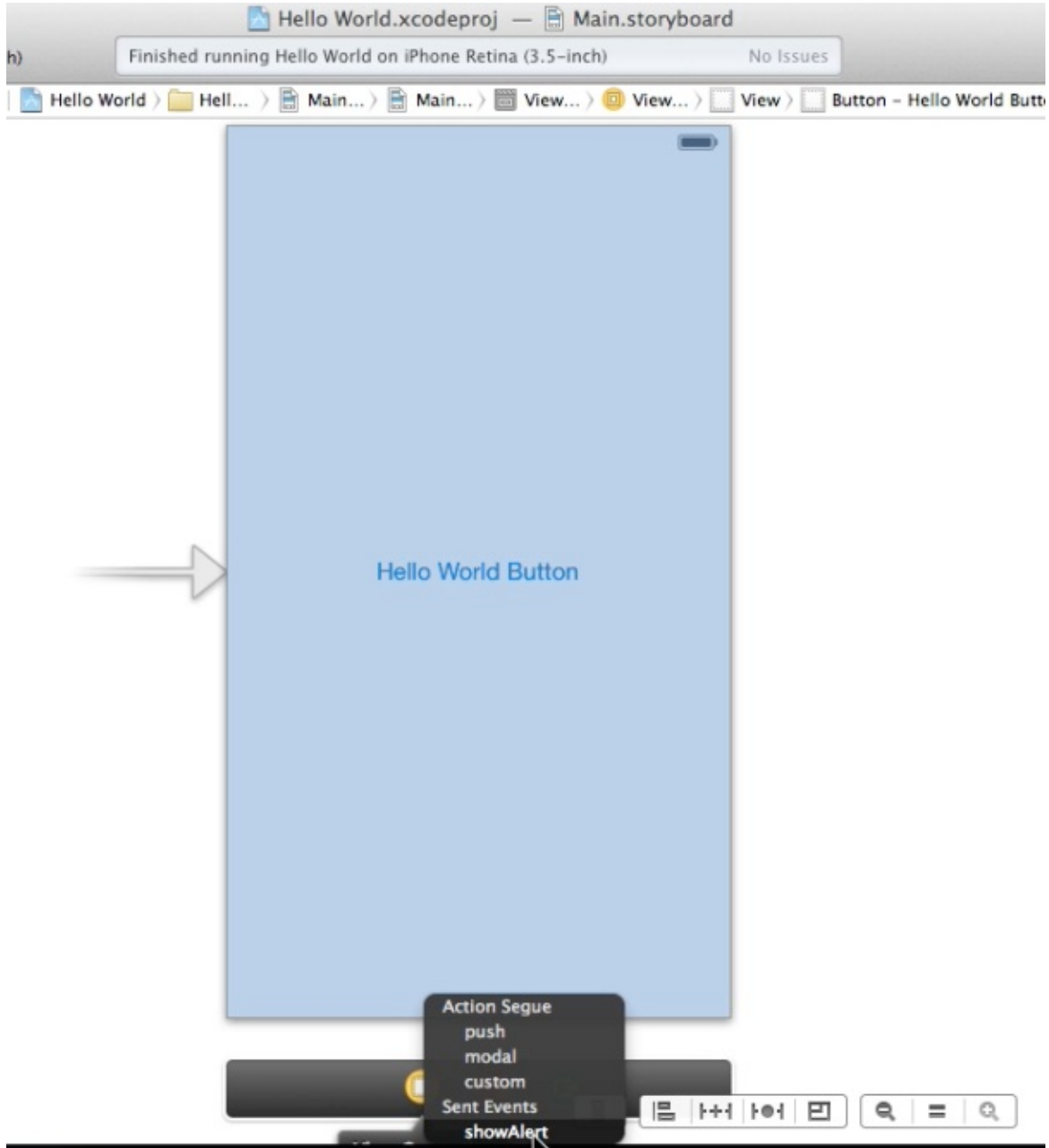
objective-c তে একটি ক্লাসের ইন্টারফেস সাধারণত *.h* ফাইল-এ লেখা হয়। অন্যদিকে *.m* ফাইল এর টেমপ্লেট কোডেও ইন্টারফেস দেখা যায় যেটিকে মূলত বলা হয় ক্লাস এক্সটেনশন। *.h* ফাইলের এর ইন্টারফেসে সেই সব প্রপার্টি, মেথড ডিক্লেয়ার করা হয় যেগুলো হয়তবা অন্য ফাইল থেকেও ব্যবহার করা হতে পারে এবং যদি এমন কিছু প্রপার্টি, মেথড প্রয়োজন হয় যেগুলো শুধুই একটি *.m* ফাইলে ব্যবহৃত হবে সেগুলোকে ক্লাস এক্সটেনশন এর মধ্যেই ডিক্লেয়ার করা ভালো অর্থাৎ ওই *.m* ফাইলের ইন্টারফেসের মধ্যে। একটি ক্লাসের নামের পূর্বে **@interface** সিনট্যাক্স ব্যবহার করে সেই ক্লাসের ইন্টারফেস ডিক্লেয়ার করা হয়।

ViewController.h ফাইলের **@interface** ও **@end** এর মধ্যে আমরাও showAlert মেথড ডিক্লেয়ার করেছি যেটাকে ফাংশন কলও বলা হয়। আর এই মেথড আসলে কি কাজ করবে সেটা আমরা লিখেছি ViewController.m ফাইলে যেখানে মূলত একটি UIAlertView তৈরী (ইনিশিয়ালাইজ) করা হয়েছে টাইটেল, ম্যাসেজ, ক্যানসেল বাটনের টেক্সট ইত্যাদি ঠিক করে দিয়ে। তারপর সেই অবজেক্ট এর show মেথড কল করে সেই এলাট কে দেখানোর ব্যবস্থা করা হয়েছে।

বাটনের কাজ ঠিক করে দেয়াঃ এখন পর্যন্ত আমাদের একমাত্র ইউজার ইন্টারফেস ফাইলে (সিনে) একটি বাটন আছে যার কোনো কাজ ঠিক করে দেয়া হয় নাই। বাম পাশের প্রজেক্ট ন্যাভিগেটর থেকে Main.storyboard সিলেক্ট করুন। আবার আমরা আমাদের ইউজার ইন্টারফেস ফাইলটি দেখতে পাচ্ছি যেখানে একটি বাটন আছে। এবার বাটনটি সিলেক্ট করে এবং কিবোর্ড এর control কি চেপে ধরে টেনে ওই সিনটির নিচে থাকা হলুদ রঙের View Controller নামের আইকনটির উপর ছেড়ে দিন। নিচের ছবির মত।

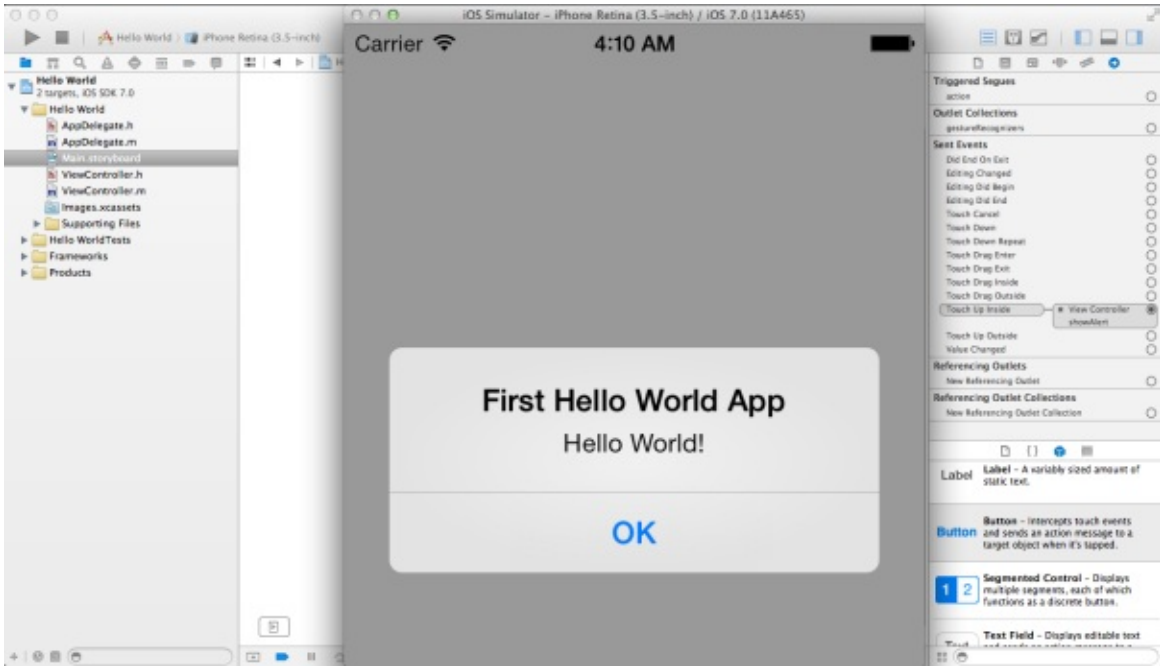


ছেড়ে দেবার পর ওখানেই ছোটো একটি পপ আপ আসবে যেখান থেকে আমাদের ফাংশন showAlert সিলেক্ট করুন। নিচের ছবির মত।



সাধারণ ভাবে, এভাবে একটি বাটনের জন্য ফাংশন ঠিক করে দিলে বাটনটির TouchUpInside ইভেন্টটির সাথে ওই ফাংশনটি জড়িয়ে যায়। অর্থাৎ যখন বাটনটি চেপে ছেড়ে দেয়া হবে তখনই ওই ফাংশনটিকে ডাকা হবে।

আমাদের কাজ শেষ। এবার আগের মত রান বাটন এ ক্লিক করুন অথবা কিবোর্ডের Command+r প্রেস করুন এপ্লিকেশনটিকে রান করার জন্য। এপ্লিকেশনটি সিমুলেটর-এ চালু হলে Hello World Button এ ক্লিক করুন। নিচের ছবির মত এলার্ট ম্যাসেজ আসলে আমরা ধরে নিচ্ছি আমাদের হ্যালো ওয়ার্ল্ড এপ্লিকেশন বানানো শেষ।



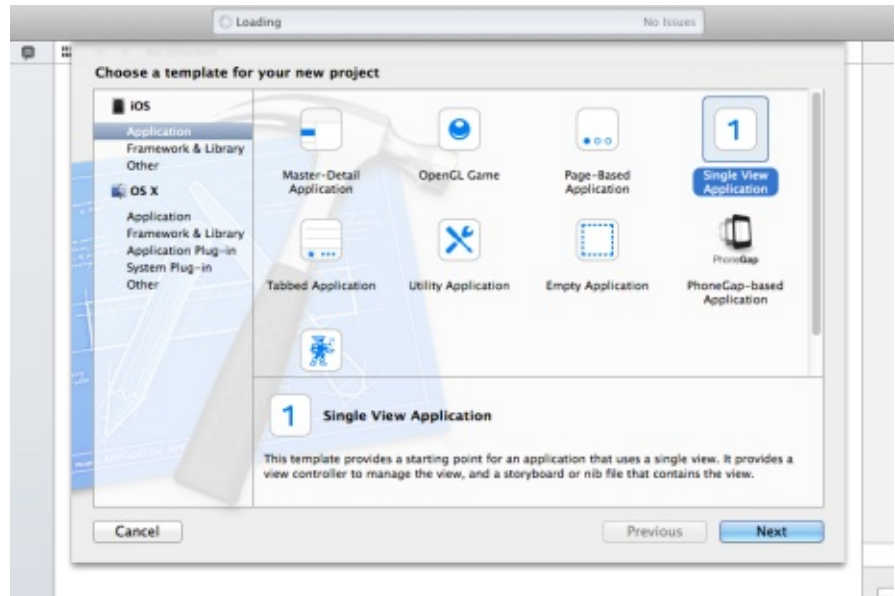
সামনে পূর্ণাঙ্গ ডাটা, এপিআই সেন্ট্রিক, সিঙ্গেল ভিউ, টেবিল ভিউ, ম্যাপ ভিউ, স্ক্রল ভিউ, মাল্টিমিডিয়া ও গেম টাইপ এপ্লিকেশন এর পুরো টিউটোরিয়াল ধাপে ধাপে প্রকাশ করা হবে। আর সেটার জন্য সংক্রিয় আপডেট পেতে হলে ইমেইল সাবস্ক্রাইব করুন এই সাইটে অথবা এ সম্পর্কিত পূর্ণাঙ্গ বই এর আপডেট লাইক দিয়ে রাখতে পারেন [এখানে](#)

Originally Posted [Here](#)

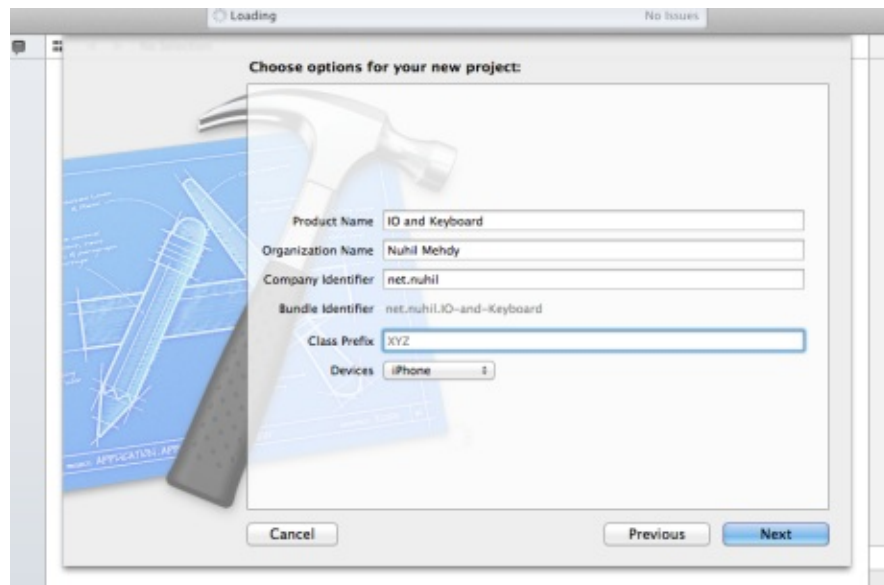
iOS অ্যাপে ব্যাসিক ইনপুট আউটপুট ও কিবোর্ড হ্যান্ডেলিং

ভূমিকাঃ এই চ্যাপ্টারে আমরা iOS অ্যাপে কিভাবে সাধারণ ডাটা ইনপুট দেয়া যায় এবং সেটা কিভাবে হ্যান্ডেল করতে হয় তা আলোচনা করবো। ডাটা ইনপুট এর সাথে সাথে যেহেতু কিবোর্ড এর ব্যাপারটাও চলে আসে তাই কিবোর্ড হ্যান্ডেলিং নিয়েও সহজ কিন্তু সবসময় কাজে লাগে এমন কিছু ফাংশনের ব্যবহার শিখবো। আর এর মাঝখানে দেখে নেব কিভাবে Apple ডকুমেন্টেশন দেখে দেখে হঠাৎ যেকোনো অচেনা এলিমেন্ট বা অবজেক্ট নিয়ে কাজ করা যেতে পারে। নিচের উদাহরণ হিসেবে আমরা সেরকম একটি অ্যাপ ধাপে ধাপে করবো যেখানে এই বিষয় গুলোর বাস্তব প্রয়োগ দেখা যাবে। চলুন শুরু করি।

প্রোজেক্ট ও ইউজার ইন্টারফেস তৈরিঃ প্রথমে Xcode ওপেন করে একটি নতুন প্রোজেক্ট তৈরি করুন File -> New -> Project... এ ক্লিক করে। এক্ষেত্রে টাইপ হিসেবে সিলেক্ট করুন iOS -> Application -> Single View

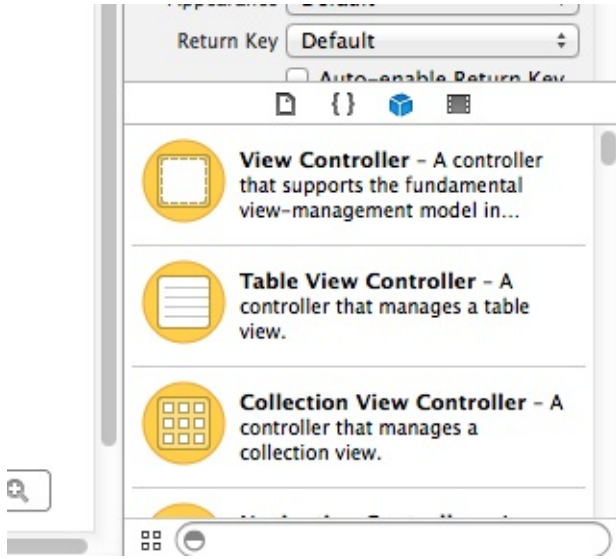


Application. নিচের মত করে,



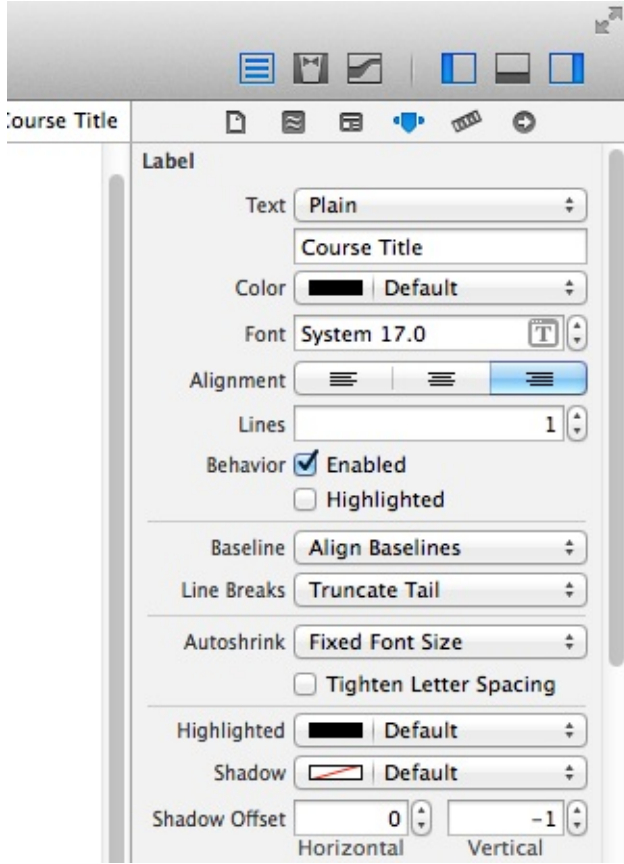
তারপরের স্ক্রিনে এভাবে,

প্রোজেক্ট তৈরি হবার পর Xcode এর বাম পাশ থেকে Main.storyboard ফাইলটি সিলেক্ট করুন। ডান পাশে একটিই মাত্র ডিউ ফাইল দেখা যাবে যেহেতু আমাদের অ্যাপ এর টাইপ সিঙ্গেল ডিউ। এরপর Xcode এর ডান পাশের নিচের ইন্টারফেস বিল্ডার প্যানেল থেকে মেইন ডিউ ফাইল বা স্ক্রিনের উপর একটি Label টাইপ এলিমেন্ট নিন,

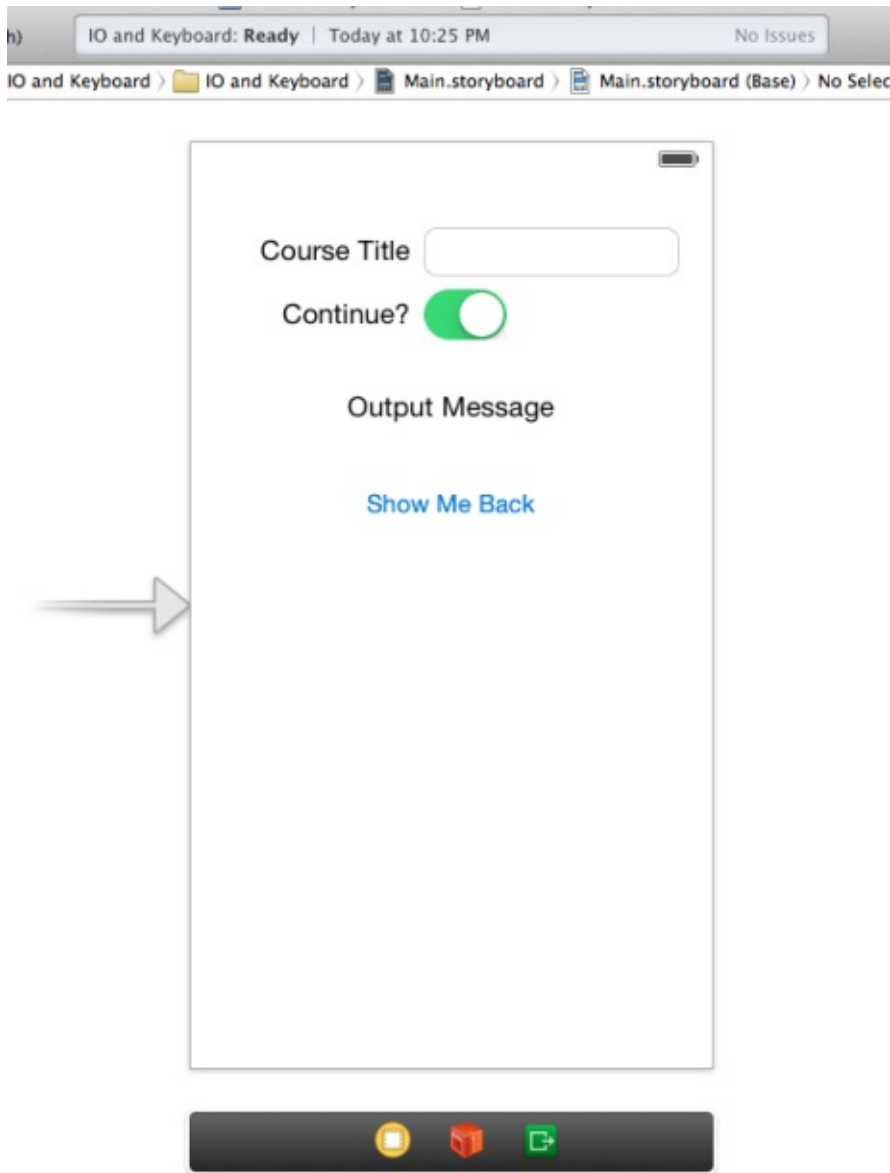


এবং সেটা সিলেক্ট থাকা অবস্থায় Xcode এর ডান পাশের

Attribute Inspector ব্যবহার করে সেটার বিভিন্ন ভিজুয়াল অ্যাপেয়ারেন্স পরিবর্তন করে নিন ইচ্ছা মত।

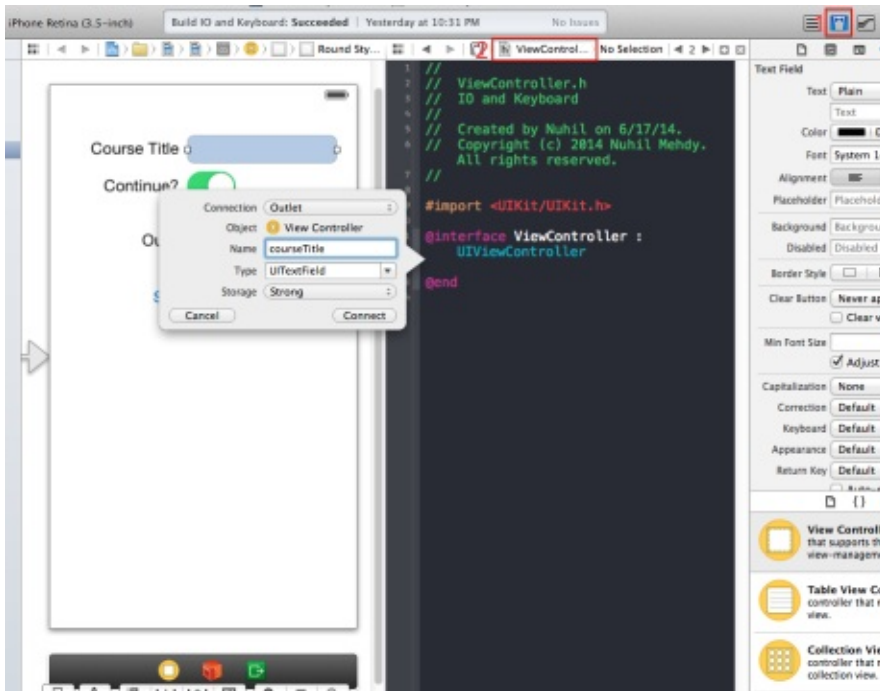


এরপর আমাদের একমাত্র ডিউ এর উপর একটি Text Field টাইপ এলিমেন্ট নিন এবং সেটারও বিভিন্ন প্রোপার্টি যেমন সাইজ, ফন্ট ইত্যাদি পরিবর্তন করে নিন ডান পাশের Attribute Inspector ব্যবহার করে। এভাবে ঠিক নিচের ছবির মত একটি লে-আউট তৈরি করে নিন। ধরে নিচ্ছি উপরে বলা কথা গুলো বুঝতে পেরেছেন। না বুঝলে এই সেকশনের [১ম চ্যাপ্টার](#) ঘুরে আসুন।



আউটলেট তৈরিঃ আমাদের ইউজার ইন্টারফেস ডিজাইন হয়ে গেছে দুটি ইউআই UI এলিমেন্ট দিয়ে। এখন এগুলোর জন্য আউটলেট তৈরি করতে হবে যাতে করে আমাদের View Controller কোডের মধ্যে থেকে এগুলোর রেফারেন্স পাওয়া যায়। এজন্য প্রথমে Xcode এর ডান দিকের উপর পাশ থেকে Assistant Editor বাটনটি এনাবেল করে নিন (নিচের ছবিতে 1 চিহ্নিত)। এনাবেল হলে ডিউ ফাইলের পাশেই আরও একটি এরিয়া তৈরি হবে যেখানে ViewController.h ফাইলটি ওপেন অবস্থায় থাকার কথা। ওই ফাইল ওপেন না থাকলে নিচের ছবিতে 2 চিহ্নিত জায়গাটায় ক্লিক করে ViewController.h ফাইলকে ওখানে ওপেন করতে পারেন। এখন কিবোর্ডের Control কি চেপে ধরে আমাদের ডিউ ফাইলের টেমপ্লেট ফিল্ডের উপর থেকে মাউস ক্লিক চেপে ধরে ডান পাশের ফাইলের

@interface এর নিচে যেকোনো জায়গায় ছেড়ে দিন। নিচের মত একটি ছোট পপ-আপ আসবে যেখানে এটার প্রোপার্টি টাইপ, নাম ইত্যাদি ঠিক করে দিতে পারবেন।



এভাবে সুইচ এলিমেন্ট, ম্যাসেজ দেখানোর লেবেল এবং বাটনটির প্রোপার্টিও ঠিক করে দিন নিচের কোডের মত,

```
// ViewController.h

#import

@interface ViewController : UIViewController
@property (strong, nonatomic) IBOutlet UITextField *courseTitle;
@property (strong, nonatomic) IBOutlet UISwitch *continueCourse;
@property (strong, nonatomic) IBOutlet UILabel *messageBox;
@property (strong, nonatomic) IBOutlet UIButton *showButton;

@end
```

কিবোর্ড হ্যান্ডেলিংঃ এখন পর্যন্ত ইন্টারফেস এলিমেন্ট গুলো এবং তাদের আউটলেট গুলো রেডি। এই অবস্থায় অ্যাপটি রান করে দেখতে পারেন। যদি সব ঠিক ঠাক ঠাকে তাহলে টেক্সট ফিল্ডটিতে কিছু লেখার জন্য ট্যাপ/ক্লিক করলে সিমুলেটরের কিবোর্ডটি চলে আসবে এবং সেটা ব্যবহার করে কিছু লিখতে পারবেন। কিন্তু লেখা শেষে দেখবেন কিবোর্ডটি সিমুলেটর থেকেই যাচ্ছে। আড়ালে চলে যাচ্ছে না। এখন আমরা কিবোর্ড হ্যান্ডেলিং এর এই সমস্যার একটা সমাধান করবো। এ জন্য প্রথমে ViewController.h ফাইলে textFieldReturn: নামের একটি ফাংশন ডিক্লেয়ার করবো। মূলত এই ফাংশনের মাধ্যমে আমরা কিবোর্ড এর রিটার্ন কি চেপে কিবোর্ডকে হাইড করে ফেলার একটা উপায় প্রয়োগ করবো।

```
// ViewController.h

#import

@interface ViewController : UIViewController
@property (strong, nonatomic) IBOutlet UITextField *courseTitle;
@property (strong, nonatomic) IBOutlet UISwitch *continueCourse;
@property (strong, nonatomic) IBOutlet UILabel *messageBox;
@property (strong, nonatomic) IBOutlet UIButton *showButton;

-(IBAction)textFieldReturn:(id)sender;

@end
```

এখন এই ফাংশনটির ইমপ্লিমেন্টেশন লিখবো ViewController.m ফাইলে নিচের মত করে,

```
// ViewController.m

#import "ViewController.h"

@interface ViewController ()

@end

@implementation ViewController

- (void)viewDidLoad
{
    [super viewDidLoad];
    // Do any additional setup after loading the view, typically from a nib.
}

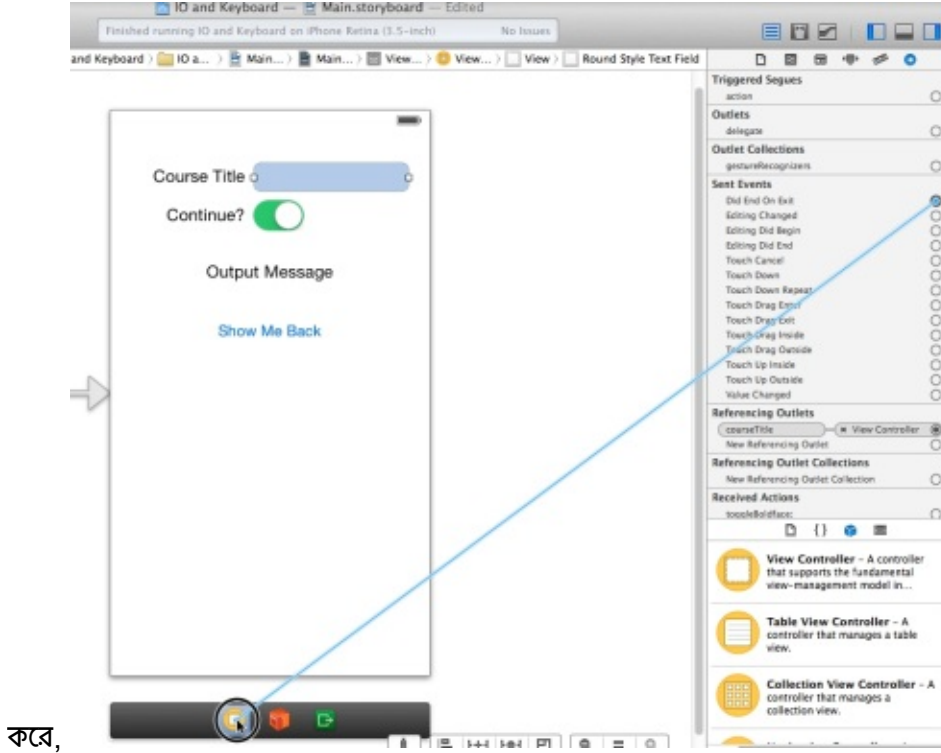
-(IBAction)textFieldReturn:(id)sender
{
    [sender resignFirstResponder];
}

- (void)didReceiveMemoryWarning
{
    [super didReceiveMemoryWarning];
    // Dispose of any resources that can be recreated.
}

@end
```

উপরের textFieldReturn: মেথডের মধ্যে থেকে আমরা যা করছি তা হল, ইভেন্টটি যে অবজেক্ট দ্বারা ট্রিগারড হয়েছে সেই অবজেক্টের resignFirstResponder মেথডকে কল করছি। First Responder হচ্ছে সেই অবজেক্ট যেটা এই মুহূর্তে ইউজারের সাথে ইন্টারঅ্যাক্ট করছে, এ ক্ষেত্রে কিবোর্ডটাই ফার্স্ট রেস্পন্ডার।

এখন `textFieldReturn:` মেথডটি যাতে সঠিক সময় কল হয় সেজন্য কিছু কাজ করতে হবে। ডিউ কন্ট্রোলার ফাইলের টেক্সট ফিল্ডটিকে সিলেক্ট করুন এবং Xcode এর ডান পাশ থেকে Connection Inspector সিলেক্ট করুন। এরপর Did End on Exit নামের সার্কেল থেকে মাউস ক্লিক করে টেনে এনে ডিউ ফাইলের নিচের View Controller আইকনের উপর ছেড়ে দিন এবং সেখান থেকে `textFieldReturn` সিলেক্ট করুন। নিচের ছবির মত



করে,

এ অবস্থায় সব সেভ করে যদি অ্যাপটি রান করেন এবং টেক্সট ফিল্ডে কিছু লিখে কিবোর্ডের Return বাটনে ক্লিক/ট্যাপ করেন তাহলে কিবোর্ডটি হাইড হয়ে যাবে, যা আমরা করতে চাচ্ছিলাম। আরও একটা সিচুয়েশনে কিবোর্ড হাইড করা ভালো ইউজার এক্সপেরিয়েন্স যেমন, টেক্সট ফিল্ড বাদে ডিউ এর অন্য কোথাও ট্যাপ করলেও যাতে কিবোর্ডটি হাইড হয়ে যায়। চলুন সেই ব্যবস্থা করি। এর জন্য আমরা `touchesBegan:` নামক ইভেন্ট হ্যান্ডেলার মেথডটি ইমপ্লিমেন্ট করবো অর্থাৎ স্ক্রিনে টাচ হলেই এটি সক্রিয় হবে। কিন্তু এই মেথডের মধ্যে আবার এটিও চেক করতে হবে যাতে কেবল মাত্র আমাদের টেক্সট ফিল্ড বাদে অন্য কোথাও টাচ হলেই কিবোর্ড হাইডের মেথড কল করতে পারি। নিচের মত করে মেথডটি `ViewController.m` ফাইলে লিখে ফেলুন। তাহলে আপডেটেড ফাইলটি হল,

```
// ViewController.m

#import "ViewController.h"

@interface ViewController ()

@end

@implementation ViewController

- (void)viewDidLoad
{
    [super viewDidLoad];
    // Do any additional setup after loading the view, typically from a nib.
}

- (IBAction)textFieldReturn:(id)sender
{
    [sender resignFirstResponder];
}

- (void)touchesBegan:(NSSet *)touches withEvent:(UIEvent *)event {

    UITouch *touch = [[event allTouches] anyObject];
    if ([self.courseTitle isFirstResponder] && [touch view] != self.courseTitle) {
        [self.courseTitle resignFirstResponder];
    }
    [super touchesBegan:touches withEvent:event];
}

- (void)didReceiveMemoryWarning
{
    [super didReceiveMemoryWarning];
    // Dispose of any resources that can be recreated.
}

@end
```

এখন আবার অ্যাপটি রান করুন এবং টেক্সট ফিল্ডে ক্লিক করুন। স্বভাবতই কিবোর্ড চলে আসবে। এখন হয় কিবোর্ডের return বাটন চাপুন নয়ত স্ক্রিনের যেকোনো জায়গায় ক্লিক/ট্যাপ করুন, কিবোর্ড হাইড হয়ে যাবে।

ইনপুট আউটপুটঃ ওকে, এবার আসুন ডাটা ইনপুট এবং সেটা স্ক্রিনে আউটপুটের ব্যবস্থা করা যাক।

ViewController.h ফাইলে নতুন একটি ফাংশন ডিক্লেয়ার করুন যাতে আপডেটেড ফাইলটি দেখতে নিচের মত হয়,

```
// ViewController.h

#import

@interface ViewController : UIViewController
@property (strong, nonatomic) IBOutlet UITextField *courseTitle;
@property (strong, nonatomic) IBOutlet UISwitch *continueCourse;
@property (strong, nonatomic) IBOutlet UILabel *messageBox;
@property (strong, nonatomic) IBOutlet UIButton *showButton;

-(IBAction)textFieldReturn:(id)sender;
-(IBAction)outputData;

@end
```

এবার এই `outputData` মেথডটির ইমপ্লিমেন্টেশন লিখে ফেলুন `ViewController.m` ফাইলে যাতে পুরো ফাইলটি দেখতে নিচের মত হয়,

```
// ViewController.m

#import "ViewController.h"

@interface ViewController ()

@end

@implementation ViewController

- (void)viewDidLoad
{
    [super viewDidLoad];
    // Do any additional setup after loading the view, typically from a nib.
}

- (IBAction)textFieldReturn:(id)sender
{
    [sender resignFirstResponder];
}

- (void)touchesBegan:(NSSet *)touches withEvent:(UIEvent *)event {

    UITouch *touch = [[event allTouches] anyObject];
    if ([self.courseTitle isFirstResponder] && [touch view] != self.courseTitle) {
        [self.courseTitle resignFirstResponder];
    }
    [super touchesBegan:touches withEvent:event];
}

- (IBAction)outputData {
    NSString *isCourseOngoing = (self.continueCourse.on)? @"Ongoing" : @"Not Ongoing";

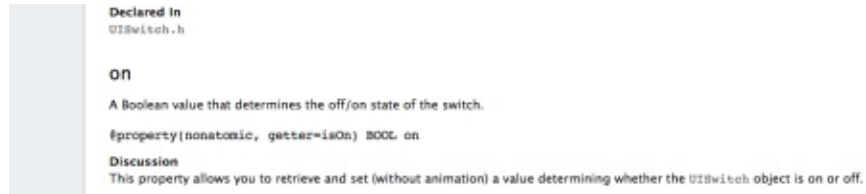
    self.messageBox.text = [NSString stringWithFormat:@"%@" is %@", self.courseTitle.text,
}

- (void)didReceiveMemoryWarning
{
    [super didReceiveMemoryWarning];
    // Dispose of any resources that can be recreated.
}

@end
```

একটি টিপস (**Apple** ডকুমেন্টেশন ফলো করা) ঃ উপরে খেয়াল করুন, outputData মেথডের মধ্যে প্রথমে আমরা isCourseOngoing -এ একটি স্ট্রিং ভ্যালু সেট করছি যেটা নির্ভর করছে continueCourse নামের আউটলেট প্রোপারটির অর্থাৎ সুইচ এলিমেন্টটির বর্তমান অবস্থার উপর। এখন কথা হচ্ছে এই যে এখানে continueCourse.on লিখে ওই অবজেক্টটির স্ট্যাটাস অ্যাক্সেস করলাম সেটা হঠাৎ আমরা জেনে নিতে পারি কই থেকে?

খেয়াল করলে দেখবেন ওই অবজেক্টটি একটি UISwitch টাইপের। অর্থাৎ এর বিস্তারিত জানা যাবে UISwitch ক্লাসের রেফারেন্স ঘেঁটে দেখলেই। আপনি ঠিকি ধরেছেন, এরকম যেকোনো ক্লাসের রেফারেন্স দেখে নিয়ে সেটার অবজেক্টের উপর আমরা বিভিন্ন অপারেশন করতে পারি। যেমন এই ক্লাসের রেফারেন্স থেকে আমরা জানতে পারি এর কি কি প্রোপার্টি ও মেথড আছে এবং সেরকম একটা প্রোপার্টি হচ্ছে on প্রোপার্টি। [এই লিঙ্কে গলেই](#) দেখতে পারবেন



লেখা আছে নিচের মত,

অর্থাৎ আমরা continueCourse.on এর মাধ্যমে ওই প্রোপার্টির ভ্যালু অ্যাক্সেস করে ঠিক করতে পারি সুইচটি কি অন নাকি অফ অবস্থায় আছে। তার ঠিক পরেই আমরা messageBox লেবেল এর text প্রোপার্টি হিসেবে একটি ফরম্যাটেড স্ট্রিং সেট করছি; courseTitle এর text প্রোপার্টি এবং isCourseOngoing এর ভ্যালু মিলিয়ে।

এবার শেষ বারের মত অ্যাপটি রান করুন আর টেক্সট ফিল্ডে যেকোনো ভ্যালু এবং তার নিচের সুচটি অন/অফ করে Show Me Back বাটনে ক্লিক করে দেখুন বাটনের ঠিক উপরের জায়গায় আপনার মন মত আউটপুট দেখাচ্ছে কিনা।

বইয়ের আপডেট পেতে চোখ রাখুন আমাদের ফ্যান পেজে

পরের চ্যাপ্টারঃ পরের চ্যাপ্টারে অটো লে-আউট নিয়ে বিস্তারিত আলোচনা থাকবে এবং তার উপর ভিত্তি করে একটি পুরো উদাহরণ থাকবে আর তার পর পরই আসবে টেবিল ডিউ নিয়ে চ্যাপ্টার।

Originally Posted [Here](#)

এই সেকশনে থাকছে

- সুইফট – অ্যাপলের নতুন চমক, পরিচিতি ও অন্যান্য
- স্ট্রিং ও ক্যারেকটার টাইপ ভ্যারিয়েবল
- কালেকশনস (অ্যারে ও ডিকশনারী), ইনুমারেশন ও ক্লোজার
- ব্রাঞ্চিং ও লুপিং
- ফাংশন ও মেথড
- ক্লাস এবং স্ট্রাকচারের মধ্যে পার্থক্য
- প্রোপার্টি
- ইনহেরিটেন্স
- ইনিশিয়ালাইজেশন, ডি-ইনিশিয়ালাইজেশন ও অপশনাল চেইনিং
- এক্সটেনশন ও প্রোটোকল

সুইফট – অ্যাপলের নতুন চমক, পরিচিতি ও অন্যান্য বেসিক

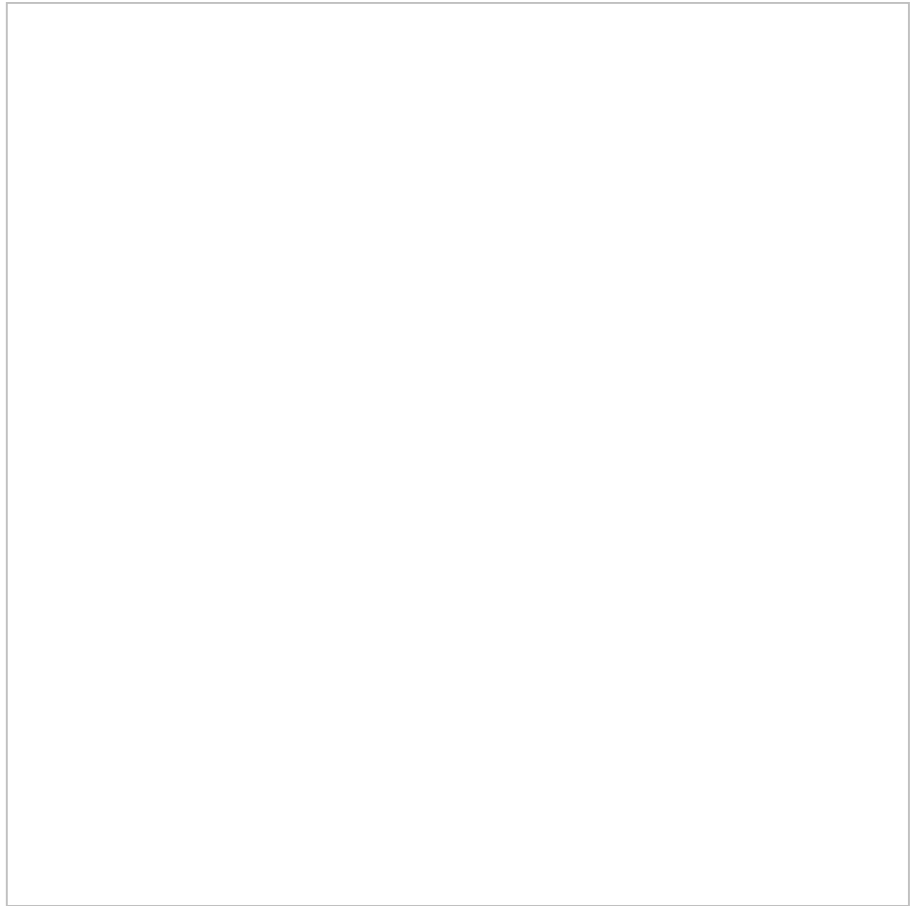
ভূমিকাঃ অবজেক্টিভ-সি (Objective-C) এর বেসিক লার্নিং (সেকশন ১) ও আইওএস জিইউআই (GUI) অ্যাপ ডেভেলপমেন্ট (সেকশন ২) এ নজর রাখার জন্য আপনাকে অসংখ্য অভিনন্দন। আমাদের সেকশন ১ লেখা শেষ হওয়ার সাথে সাথেই অনুষ্ঠিত হয় অ্যাপলের ওয়ার্ল্ড ওয়াইড ডেভেলপার কনফারেন্স - ২০১৪ (WWDC - 2014)। প্রতিবছরের মত এবারও অ্যাপলের ওয়ার্ল্ড ওয়াইড ডেভেলপার কনফারেন্স - ২০১৪ (WWDC) -এ ডেভেলপাররা পেয়েছে ৩ টি নতুন চমক। একদিকে আইওএস-৮ (iOS 8) ও ইউসেমাইট (Yosemite, OS X 10.1০) এবং অন্যদিকে সুইফট (Swift)। সুইফট হল একেবারেই নতুন একটি প্রোগ্রামিং ল্যাঙ্গুয়েজ যা দিয়ে অবজেক্টিভ-সি এর মতই আইফোন, আইপ্যাড ও আইপডের জন্য অ্যাপ্লিকেশন ডেভেলপ করা যাবে। প্রায় ৩০ বছর ধরে অবজেক্টিভ-সি একাই রাজত্ব করে আসছে আইওএস অ্যাপ্লিকেশন ডেভেলপমেন্ট ফিল্ডে। অ্যাপলের মতে অদূর ভবিষ্যতে অবজেক্টিভ-সি কে সরিয়ে এই রাজত্ব দখল করে নেবে সুইফট নিজের ফাস্ট (first), মডার্ন (modern), সেইফ (safe), ইন্টারঅ্যাক্টিভ (interactive) বৈশিষ্ট্যের দ্বারা।

সুইফট - প্রোগ্রামিং ল্যাঙ্গুয়েজ :ঃ আপনার যদি ওয়েব ডেভেলপমেন্ট (জাভাস্ক্রিপ্ট ও অন্যান্য স্ক্রিপ্টিং ল্যাঙ্গুয়েজ) সম্পর্কে ধারণা থাকে, তাহলে সুইফট এর সিনট্যাক্স আপনার কাছে পরিচিত মনে হবে এবং আপনি অনেক সহজেই সুইফটের সিনট্যাক্স আয়ত্ত্ব করতে পারবেন।

কিছু বিশেষ অবস্থার প্রিভিউ দেখার জন্য, সুইফট লেখা প্রোগ্রাম কোন সিমুলেটর এ কম্পাইল ও রান করানো লাগবে না। Xcode 6 এ প্লেগ্ৰাউন্ড (playground) নামক নতুন ফিচার পাবেন যা দিয়ে সহজে সুইফট এ লেখা প্রোগ্রামের রিয়েল টাইম আউটপুট দেখা যাবে।

মাত্রই ঘোষিত হওয়া সুইফট শেখার জন্য অ্যাপল দিয়েছে পর্যাপ্ত রিসোর্স। একটি পিডিএফ ফরম্যাটের বই (অ্যাপলের দেওয়া সুইফট এর ফ্রি বই) এবং আরও আছে কিছু অনলাইন রিসোর্স।

সুইফট ল্যাঙ্গুয়েজে হ্যালো ওয়ার্ল্ড প্রোগ্রামঃ XCode 6 এ একটি নতুন প্রজেক্ট ওপেন করি। যেভাবে অবজেক্টিভ-সি ও আইফোন ডেভেলপমেন্ট প্রজেক্ট ওপেন করা হয়, সুইফট এর জন্য একিভাবে প্রজেক্ট ওপেন করতে হয়। শুধুমাত্র প্রথমে Playgroud টাইপ প্রজেক্ট ওপেন করা এবং ল্যাঙ্গুয়েজ লিস্ট থেকে "swift" সিলেক্ট করতে হবে। এই সুবিধাটি



XCode 6 থেকে পাওয়া যাবে।

তাহলে চলুন এবার সুইফট ল্যান্সুয়েজে হ্যালো ওয়ার্ল্ড প্রোগ্রামটি লিখে ফেলিঃ

```
println("Hello, World")
```

সদ্য তৈরি করা প্রজেক্টের মূল ফাইলটিতে শুধুমাত্র এটুকু লিখেই রান করলে ডানদিকের আউটপুট উইন্ডোতে "Hello, world" দেখা যাবে।

লাইব্রেরী ইমপোর্ট (**Import**) করা সুইফট ল্যান্সুয়েজ দিয়ে করা প্রোগ্রামেও যদি কোন এক্সটার্নাল লাইব্রেরী প্রয়োজন হয় তাহলে প্রজেক্টটিতে ওই লাইব্রেরী ইমপোর্ট করতে হয়। লাইব্রেরী ইমপোর্ট করার জন্য শুধুমাত্র import দিয়ে লাইব্রেরীর নাম লিখলেই হয়।

```
import Foundation

println("Foundation Library is imported.");
```

ভ্যারিয়েবল, কনস্ট্যান্ট ও টাইপ অ্যানোটেশনঃ অবজেক্টিভ-সি (Objective-C) এর মত ভ্যারিয়েবল বা কনস্ট্যান্ট ডিক্লেয়ার করার জন্য সুইফট এ আসল ডাটাটাইপ লেখা লাগে না। সুইফট এ ভ্যারিয়েবল ডিক্লেয়ার করার জন্য শুধুমাত্র "var" ও কনস্ট্যান্ট ডিক্লেয়ার করার জন্য "let" ব্যবহার করা হয়। সুবিধা হল ভ্যারিয়েবল বা কনস্ট্যান্ট ডিক্লেয়ার করার সময় এর ডাটাটাইপ সম্পর্কে মাথা ঘামানোর প্রয়োজন নেই। যে ধরনের ডাটা দিয়ে ইনিশিয়ালাইজ করা হবে, এদের ডাটাটাইপ ও তাই হবে। নিজের প্রোগ্রামটুকু দেখলেই পরিষ্কার হয়ে যাবে।

```
println("Hello, world")
var myVariable = 42
// myVariable is a variable which will be inferred to be int

var (x, y, z) = (11, 12, 45) // x = 11, y = 12, z = 45
// x, y and z variables are inferred to be int

let π = 3.1415926 // constant
// π is inferred to be int

let (x, y) = (10, 20.8) // x = 10, y = 20.8
// x is inferred to be int and y is inferred to be double
```

উপরের কোড থেকে দেখা যাচ্ছে যে parentheses "(")" ব্যবহার করে একসাথে একাধিক ভ্যারিয়েবল বা একাধিক কনস্ট্যান্ট ইনিশিয়ালাইজ করা যায়। আবার কনস্ট্যান্ট এর নাম হিসেবে "π" লেখা হয়েছে। সুইফট ভ্যারিয়েবল বা কনস্ট্যান্ট অথবা অন্যান্য আইডেন্টিফায়ারের নাম গুলোতে যেকোন স্পেশাল ক্যারেকটার এমনকি ইমোজি (emoji) আইকন ও ব্যবহার করা যায়। উদাহরণস্বরূপ,

```
let π = 3.14159 // any special characters
let 你好 = "你好世界" // any unicode supported characters
let সিঁড়ি = "একজন কিংবদন্তীর নাম" // any unicode supported characters
let 🐶 = "dogcow" // any emoji icons
```

emoji ক্যারেকটারের লিস্ট পাওয়ার জন্য কন্ট্রোল + কমান্ড + স্পেসবার একসাথে চাপ দিন।

আপনি চাইলে ভ্যারিয়েবল বা কনস্ট্যান্ট ডিক্লেয়ার করার সময় এক্সপ্লিসিটলি ডাটাইপ জানিয়ে দিতে পারেন। এজন্য ভ্যারিয়েবল বা কনস্ট্যান্ট এর নামের পর কোলন (:) ও তারপর স্পেস দিয়ে ডাটাইপ লিখতে হয়। এভাবে আলাদা করে ডাটাইপ সম্পর্কে জানানো কে বলা হয় টাইপ অ্যানোটেশন (Type Annotation)।

```
var annotatedMessage: String = "Hello World! This variable is Type Annotated"
```

আচ্ছা, আমরা কি খেয়াল করেছি যে, উপরের প্রোগ্রামগুলোর কোন স্টেটমেন্টের শেষে সেমিকোলন নেই। অন্যান্য প্রোগ্রামিং ল্যাঙ্গুয়েজের মত সুইফট স্টেটমেন্টের শেষে সেমিকোলন লাগে না। কিন্তু আপনার যদি ইচ্ছা হয় তাহলে দিতে পারেন। এতে কোন রকম এরর হবে না। আবার যদি আপনি একটি লাইনে একাধিক স্টেটমেন্ট লিখতে চান সেক্ষেত্রে অবশ্যই স্টেটমেন্টগুলোর শেষে সেমিকোলন দিতে হবে।

```
println("I am the first statement. I need semicolon"); println("I am the last statement.
println("Another Line, I don't need any semicolon. But you can give me a semicolon if you
```

অপারেটরঃ অন্যান্য প্রোগ্রামিং ল্যাঙ্গুয়েজের অপারেটর গুলোর সাথে সুইফট এর অপারেটর গুলোর মধ্যে তেমন কোন পার্থক্য নেই। তবে কিছু কিছু ক্ষেত্রে অনেকটা ইমপ্রুভমেন্ট এনেছে সুইফট। নিচে ক্যাটেগরী অনুযায়ী কিছু অপারেটর সম্পর্কে ধারণা দেওয়া হলঃ

অ্যাসাইনমেন্ট অপারেটর (=) অ্যাসাইনমেন্ট অপারেটর (=) একটি বাইনারি অপারেটর। অর্থাৎ "=" এর দুইটি অপারেণ্ড (Operands) থাকবে এবং ডানদিকের ভ্যালু বামদিকের ভ্যারিয়েবল বা কনস্ট্যান্ট এ অ্যাসাইন হবে।

```
let a = 5
let b = 10
var c = a + b
var (x, y) = (a, b) // a is assigned to x and b is assigned to y
var httpNotFoundError = (404, "Page Not Found")
```

শেষ লাইন দুটি দেখে বোঝা যাচ্ছে যে, এভাবে "=" অপারেটরের ডানদিকে যদি একাধিক ডাটা থাকে তাহলে তা বামদিকের একাধিক সংখ্যক ভ্যারিয়েবল বা কনস্ট্যান্ট এ অ্যাসাইন হতে পারে। আবার কোন একটি ভ্যারিয়েবল বা কনস্ট্যান্ট এর ভ্যালু একাধিক ভ্যালুর টাপল [var a = (b,c)] হতে পারে।

অ্যারাথমেটিক অপারেটর অন্যান্য সকল প্রোগ্রামিং ল্যাঙ্গুয়েজের অ্যারাথমেটিক অপারেটর গুলোর মতই সুইফট এর অ্যারাথমেটিক অপারেটর গুলো নিজ নিজ অপারেশন করে।

```
1 + 2 // equals 3 (Addition)
5 - 3 // equals 2 (Subtraction)
2 * 3 // equals 6 (Multiplication)
10.0 / 2.5 // equals 4.0 (Division)
9 % 4 // equals 1 (Remainder after division)
-9 % 4 // equals -1 (Remainder after division)
8 % 2.5 // equals 0.5
```

"+" অপারেটর দিয়ে স্ট্রিং কনক্যাটেনেশনও করা যায়।

```
var msg1 = "Hello"
var msg2 = "World"
var concatenated = msg1 + msg2
println(concatenated) // Hello World
```

অন্যান্য অ্যারাথমেটিক অপারেটর

```

var a = 10;
++a      // equals 11
a++      // equals 11

// currently a is 11
var b = a++ // b = 11, but a = 12
// at first a is assigned to b. So, value of b is same as current a (11). Then a is incre

//currently a is 12
var b = ++a // b = 13, and a = 13
// at first a is incremented by 1. So, value of a is 13 (12+1). Then value of a is assign

//currently a is 13
var b = a-- // b = 13, but a = 12
// at first a is assigned to b. So, value of b is same as current a (13). Then a is decre

//currently a is 11
var b = --a // b = 11, and a = 11
// at first a is decremented by 1. So, value of a is 11 (12-1). Then value of a is assign

```

কমপ্যারিজন/কন্ডিশনাল অপারেটর সুইফট সকল ধরনের কন্ডিশনাল অপারেটর গুলো সাপোর্ট করে। নিচে কয়েকটি অপারেটরের নাম দেওয়া হলঃ

- Equal to (a == b)
- Not equal to (a != b)
- Greater than (a > b)
- Less than (a < b)
- Greater than or equal to (a >= b)
- Less than or equal to (a <= b)

```

1 == 1 // true, because 1 is equal to 1
2 != 1 // true, because 2 is not equal to 1
2 > 1 // true, because 2 is greater than 1
1 < 2 // true, because 1 is less than 2
1 >= 1 // true, because 1 is greater than or equal to 1
2

```

এধরনের অপারেটর গুলো প্রোগ্রামের বিভিন্ন সিলেকশন বা ডিসিশন নেওয়ার সময় ব্যবহৃত হয়। পরবর্তী অধ্যায়সমূহে এ ব্যাপারে বিস্তারিত আলোচনা হবে।

এছাড়াও রয়েছে লজিক্যাল অপারেটর ও রেন্জ অপারেটর। সকল অপারেটর সম্পর্কে আরও বিস্তারিত আলোচনা ও বিশ্লেষণ থাকবে আমাদের আসছে প্রিন্টেড বইয়ে।

বইয়ের আপডেট পেতে চোখ রাখুন আমাদের ফ্যান পেজে

পরের চাপ্টারঃ পরের চাপ্টারে সুইফট ল্যান্ডুয়েজ দিয়ে স্ট্রিং ও ক্যারেকটার এর ম্যানিপুলেশন সম্পর্কে বিস্তারিত থাকবে।

এছাড়া কিভাবে স্ট্রিং কে মিউটেবল করা হয়, কিভাবে একটি স্ট্রিং থেকে নির্দিষ্ট ইনডেক্সের ক্যারেকটার খুঁজতে হয়, কিভাবে ইন্টারপুলেশন, কমপ্যারিজন করতে হয় ইত্যাদি সম্পর্কে থাকবে বিস্তারিত।

Originally Posted [Here](#)

স্ট্রিং ও ক্যারেকটার টাইপ ভ্যারিয়েবল

ভূমিকাঃ অ্যাপলের নতুন প্রোগ্রামিং ল্যাঙ্গুয়েজ সুইফট নিয়ে লেখা আমাদের তৃতীয় সেকশনের প্রথম অধ্যায়ে সুইফট সম্পর্কে পরিচিতি মূলক আলোচনা হয়েছে। আমরা সুইফট ল্যাঙ্গুয়েজের বেসিক সিনট্যাক্স, ভ্যারিয়েবল ও কনস্ট্যান্ট ডিক্লেয়ার করা, বিভিন্ন ধরনের অপারেটর সম্পর্কে জেনেছি। এই অধ্যায়ে ক্যারেকটার ও স্ট্রিং ম্যানিপুলেশনের বিস্তারিত থাকবে।

ক্যারেকটার ও স্ট্রিং ঃ অন্যান্য বেসিক ডাটাতাইপ (Integer, Float, Double) গুলোর মতই ক্যারেকটার (Character) একটি ডাটাতাইপ যা এক বাইট (Byte) ডাটা সংরক্ষণ করে। অন্যদিকে ক্যারেকটার টাইপ অ্যারে কে স্ট্রিং(String) বলা হয়। অর্থাৎ যদি একাধিক ক্যারেকটার একত্রে কোন নির্দিষ্ট নিয়মে সাজানো হয় তাহলে এই ক্যারেকটারগুলোকে একত্রে একটি স্ট্রিং বলা হয়। যেমন ঃ "Steve Jobs", "Swift", "Programming", "Macbook Pro and My iPhone" ইত্যাদি। অনেকসময় একটি ক্যারেকটার কে ও স্ট্রিং হিসেবে ব্যবহার করা যায় যার উদাহরণ আমরা একটু পরেই দেখব।

অন্যান্য প্রোগ্রামিং ল্যাঙ্গুয়েজের চেয়ে সুইফট এ স্ট্রিং ও ক্যারেকটার ম্যানিপুলেট করা অনেক বেশী ফাস্টার এবং সুইফট ক্যারেকটার গুলোতে ইউনিকোড ক্যারেকটার স্টোর করে। অর্থাৎ যেকোন ক্যারেকটারে ভ্যালু হিসেবে ইউনিকোড স্টোর করা যায় যা এনকোডিং ডিকোডিং এর ঝামেলা কমিয়ে দিয়েছে। এছাড়া সুইফট ল্যাঙ্গুয়েজে স্ট্রিং ম্যানিপুলেশন খুবই সহজ। খুব সহজেই যেকোন স্ট্রিং এর কোন ইন্ডেক্সে কনস্ট্যান্ট, ক্যারেকটার, নাম্বার, এক্সপ্রেশন ইত্যাদি ইনসার্ট করা যায়। এভাবে স্ট্রিং এর মধ্যে কোন কিছু ইনসার্ট করাকে ইন্টারপুলেশন বলা হয়।

স্ট্রিং ডিক্লেয়ার করাঃ কিভাবে ভ্যারিয়েবল ও কনস্ট্যান্ট ডিক্লেয়ার করতে হয় তা আমরা প্রথম অধ্যায়ে শিখেছি। একই ভাবে স্ট্রিং টাইপ ভ্যারিয়েবল বা কনস্ট্যান্ট ডিক্লেয়ার করতে হয়।

```
let stringValue = "যেকোন ইউনিকোড মাপোর্টেড ক্যারেকটারসমূহ"
let anotherStringVariable: String = "Yosemite is the name of latest OS X"
```

যেহেতু stringValue ও anotherStringVariable দুটিতে স্ট্রিং দিয়ে ইনিশিয়ালাইজ করা হয়েছে তাই এই কনস্ট্যান্ট দুটি এখন স্ট্রিং টাইপ কনস্ট্যান্ট হিসেবে কাজ করবে। এবং এদের উপর যাবতীয় স্ট্রিং অপারেশনগুলো করা যাবে।

স্ট্রিং এ ক্যারেকটার ছাড়াও অন্যান্য যা যা থাকতে পারে ঃ

- সব ধরনের স্পেশাল (escaped special) ক্যারেকটার যেমন ঃ \0 (null), \n (new line), \ (backslash), \t (horizontal tab), \" (double quote), ' (single quote) ইত্যাদি।
- এক বাইটের ইউনিকোড স্কেলার (\xnn, nn এর জায়গায় যেকোন দুটি হেক্সাডেসিমেল ডিজিট বসতে পারে)
- দুই বাইটের ইউনিকোড স্কেলার (\xnnnn, nnnn এর জায়গায় যেকোন চারটি হেক্সাডেসিমেল ডিজিট বসতে পারে)
- চার বাইটের ইউনিকোড স্কেলার (\xnnnnnnnn, nnnnnnnn এর জায়গায় যেকোন আটটি হেক্সাডেসিমেল ডিজিট বসতে পারে)

এই চারধরনের বিশেষ ক্যারেকটার গুলো নিয়ে নিচে চারটি উদাহরণ দেওয়া হলঃ


```
let wiseWords = "\"Imagination is more important than knowledge\" - Einstein"
// "Imagination is more important than knowledge" - Einstein
let dollarSign = "\x24"           // $, Two Byte Unicode scalar U+0024
let blackHeart = "\u2665"         // ♥, Four Byte Unicode scalar U+2665
let sparklingHeart = "\U0001F496" // , Eight Byte Unicode scalar U+1F496
```

ফাঁকা স্ট্রিং ইনিশিয়ালাইজ করাঃ সাধারণত প্রোগ্রামের শুরুতে ফাঁকা স্ট্রিং ইনিশিয়ালাইজ করা হয় যা পরবর্তীতে বিভিন্ন সময়ে বিভিন্ন অপারেশনের মাধ্যমে বড় সাইজের ক্যারেকটারের কালেকশন হয়ে কোন অর্থ বা তথ্য বহন করে। দুইভাবে এরকম ফাঁকা স্ট্রিং ইনিশিয়ালাইজ করা যায়। ড্যারিয়েবল বা কনস্ট্যান্টে সরাসরি কোন ফাঁকা স্ট্রিং অ্যাসাইন করে অথবা String টাইপের নতুন ইনস্ট্যান্স ইনিশিয়ালাইজ করে ফাঁকা স্ট্রিং ইনিশিয়ালাইজ করা হয়। নিচে দুই ধরনের পদ্ধতির উদাহরণ দেওয়া হলঃ

```
var ফাঁকাস্ট্রিং = ""                // empty string literal
var emptyString = ""
var anotherEmptyString = String() // initialiser syntax
// these three strings are all empty, and are equivalent to each other
```

প্রয়োজনে খুব সহজেই isEmpty প্রপার্টির বুলিয়ান ভ্যালু দেখে কোন স্ট্রিং ফাঁকা কিনা তা চেক করা যায়।

```
var emptyString = ""
if emptyString.isEmpty {
    println("Nothing to see here")
}
// prints "Nothing to see here"
```

স্ট্রিং মিউট্যাবিলিটি (String Mutability) ঃ কোন স্ট্রিং মিউট্যাবল(Mutable or Modifiable) হবে কিনা তা নির্ধারিত হয় ড্যারিয়েবল বা কনস্ট্যান্ট ডিক্লেয়ার করার সময়। যদি স্ট্রিংটি "var" টাইপ দিয়ে ডিক্লেয়ার করা হয় তাহলে এটিকে মডিফাই করা যাবে। কিন্তু যদি এই স্ট্রিংটি "let" টাইপ হয় তাহলে এটি একটি কনস্ট্যান্ট এর ন্যায় আচরণ করবে। তাই "let" টাইপ স্ট্রিং মিউট্যাবল না। নিচের উদাহরণ টি দেখলেই পরিষ্কার হয়ে যাবে ব্যাপার টি।

```
var variableString = "Swift is"
variableString += " a new programming language."
// variableString is now "Swift is a new programming language."

let constantString = "Swift is very much"
constantString += " faster and interactive"
// this reports a compile-time error - a constant string cannot be modified
```

নোটঃ সুইফট (Swift) এর এই স্ট্রিং এর সাথে অবজেকটিভ-সি (Objective-C) এর NSString রয়েছে ব্রিজ কানেকশন। আমরা যদি Cocoa অথবা Cocoa Touch এর Foundation ক্লাস নিয়ে কাজ করি তাহলে সুইফট এর স্ট্রিং টাইপ ড্যারিয়েবলগুলোর জন্য NSString ক্লাসের সব এপিআই (API) কল করা যায়।

Cocoa বা Cocoa Touch এর Foundation ক্লাসের NSString এর ইনস্ট্যান্স তৈরী করে যদি ফাংশন বা মেথডে পাঠানো হয় তাহলে মূলত ওই স্ট্রিং এর রেফারেন্স বা পয়েন্টারকেই পাঠানো হয় যা নতুন পয়েন্টার বা রেফারেন্সে অ্যাসাইন করা হয়। আসল স্ট্রিং এর কোন কপি তৈরী হয় না। অন্যদিকে Swift এর String ড্যালাু তৈরী করে যদি কোন ডারিয়েবলে অ্যাসাইন করা হয় অথবা কোন মেথড বা ফাংশনে পাঠানো তাহলে প্রথমে ওই স্ট্রিং এর একটি কপি তৈরী হয় এবং তারপর এই নতুন স্ট্রিং টি মেথডে পাঠানো হয় অথবা নতুন ডারিয়েবলে অ্যাসাইন করা হয়। আমাদের কাণ্ডজে বইতে এই সম্পর্কে আরও বিস্তারিত থাকবে।

ক্যারেকটার ম্যানিপুলেশনঃ এতক্ষনে আমরা জেনে গেছি যে, স্ট্রিং আসলে একত্রে থাকা অনেকগুলো Character যা সুইফ্ট এ String টাইপ দিয়ে রিপ্রেজেন্ট করা হয়। প্রত্যেকটি Character আবার একটি ইউনিকোড ক্যারেকটার। সুইফ্টে for-in লুপ দিয়ে একটি স্ট্রিং এর প্রত্যেকটি ক্যারেকটার এক্সেস করা যায়। for-in লুপ এর সিনট্যাক্স ও বিস্তারিত পরের অধ্যায়ে থাকবে। এই মুহুর্তে আমরা একটি স্ট্রিং এর প্রথম থেকে শেষ পর্যন্ত সবগুলো Character এক্সেস করব এবং তা দেখব।

```
for ch in "Macintosh" {
    println(ch)
}
// M
// a
// c
// i
// n
// t
// o
// s
// h
```

আবার কোন এক ক্যারেকটারের স্ট্রিং থেকে Character টাইপ ডারিয়েবল বা কনস্ট্যান্ট ডিক্লেয়ার করার জন্য নিচের মত কোড লিখতে হয়।

```
let charConstant: Character = "$"
var charVariable: Character = "ক"
```

স্ট্রিং এ থাকা মোট ক্যারেকটারের সংখ্যা জানাঃ countElements() একটি গ্লোবাল মেথড যা আর্গুমেন্ট হিসেবে একটি স্ট্রিং নেয় এবং এই স্ট্রিং এ কয়টি ক্যারেকটার আছে তা রিটার্ন করে।

```
let intro: String = "In Swift, It is too easy to count characters."
var countChar = countElements(intro);
println(countChar) // 45
```

নোটঃ সুইফ্ট এর স্ট্রিং এর ক্যারেকটারগুলো ইউনিকোড ক্যারেকটার হয় এবং ইউনিকোডে বিভিন্ন ক্যারেকটারের সাইজ ও আলাদা হয়। একারণে সুইফ্ট ল্যাঙ্গুয়েজে সকল ক্যারেকটার একই সাইজের মেমরী নেয় না। তাই countElements() মেথডের রিটার্ন ড্যালাু ও আমাদের চাওয়া অনুযায়ী হয় না। তাই কোন স্ট্রিং এ ক্যারেকটারের সঠিক সংখ্যা জানার জন্য লুপ ব্যবহার করে প্রত্যেকটি ক্যারেকটার গননা করতে হয়। নিচের উদাহরণটিতে প্রথম

লাইনে counter ভ্যারিয়েবল ০ দিয়ে ইনিশিয়ালাইজ করা হয়েছে। এরপর for-in লুপ দিয়ে প্রথম থেকে শেষ পর্যন্ত প্রত্যেকটি ক্যারেকটার ডিজিট করা হয় এবং প্রতিবার counter এর ভ্যালু ১ করে বাড়ানো হয়। ফলে for-in টির এক্সিকিউশন শেষ হলে counter ভ্যারিয়েবলে "Macintosh" এর মোট ক্যারেকটারের সংখ্যা পাওয়া যাবে।

```
var counter = 0
for ch in "Macintosh" {
    counter++;
}
println("This string has \(counter) characters"); // This string has 9 characters
```

স্ট্রিং এবং ক্যারেকটার কনক্যাট (**Concate**) করা :ঃ স্ট্রিং ও ক্যারেকটার টাইপ ভ্যারিয়েবল বা কনস্ট্যান্ট একত্রে যুক্ত (Concatenation) করে নতুন স্ট্রিং ইনিশিয়ালাইজ করা যায়। আবার কোন স্ট্রিং এর সাথে ক্যারেকটার যুক্ত (Concate) করে স্ট্রিং টিকে মডিফাই (Modify) করা যায়। নিচের উদাহরন গুলো দেখলেই ব্যপারগুলো বোঝা যাবে।

```
//Example 1
let string1 = "hello"
let string2 = " Steve"
let character1: Character = "!"
let character2: Character = "?"

let stringPlusCharacter = string1 + character1 // equals "hello!"
let stringPlusString = string1 + string2 // equals "hello Steve"
let characterPlusString = character1 + string1 // equals "!hello"
let characterPlusCharacter = character1 + character2 // equals "!"

//Example 2
var instruction = "follow"
instruction += string2
// instruction now equals "follow Steve"

var welcome = "good morning"
welcome += character1
// welcome now equals "good morning!"
```

স্ট্রিং ইন্টারপুলেশন (**String Interpolation**) :ঃ কোন স্ট্রিং এর মধ্যে নতুন করে কোন কনস্ট্যান্ট, ভ্যারিয়েবল, এক্সপ্রেশন ইত্যাদির মিশিয়ে নতুন কোন স্ট্রিং তৈরী করা কে বলা হয় স্ট্রিং ইন্টারপুলেশন। এজন্য যে নতুন ভ্যারিয়েবল বা কনস্ট্যান্ট বা এক্সপ্রেশন মেশানো হবে তা প্যারেনথেসিস বা "(" এর ভিতর লিখে তার আগে একটি ব্যাকস্ল্যাশ দিতে হয়। তাহলে স্ট্রিং এই অংশে নতুন জিনিস টি ইনসার্ট হয়ে যায়।

```
let multiplier = 3
let message = "\multiplier) times 2.5 is \Double(multiplier) * 2.5)"
// message is "3 times 2.5 is 7.5"
```

উপরের কোডটিতে multiplier কনস্ট্যান্টটি ২ বার ব্যবহার করে একটি নতুন স্ট্রিং message তৈরী করা হয়েছে। এজন্য যেসব পজিশনে multiplier এর ভ্যালু ইনসার্ট করা হবে সেসব স্থানে প্রথমে ব্যাকস্ল্যাশ ও তারপর প্যারেনথেসিস দিয়ে multiplier লেখা হয়েছে। ইন্টারপুলেশনের সময় "(multiplier)" এর পরিবর্তে এখানে multiplier এর ভ্যালু তথা 3 বসবে। এখানে লক্ষ্য করার বিষয় হল প্রথম multiplier টি একটি কনস্ট্যান্ট যা সরাসরি স্ট্রিং টিতে ইনসার্ট করা হয়েছে। কিন্তু ২য় টিতে একটি এক্সপ্রেশন "Double(multiplier) * 2.5" ইনসার্ট করা হয়েছে। প্রথমে multiplier এর ভ্যালু Double দিয়ে কাস্ট (Cast) করে তারপর 2.5 দিয়ে গুন করার পর যে রেসাল্ট পাওয়া যাবে তাই ইনসার্ট করা হয়েছে স্ট্রিং টিতে।

স্ট্রিং কমপ্যারিজন (**String Comparison**)ঃ সুইফট (Swift) এর String টাইপের মধ্যকার কমপ্যারিজন করার জন্য ৩টি মেথড রয়েছে : স্ট্রিং(String) ইকুয়্যালিটি, প্রিফিক্স(Prefix) ইকুয়্যালিটি এবং সাফিক্স(Suffix) ইকুয়্যালিটি।

২ টি স্ট্রিং ইকুয়্যাল হবে যদি এবং কেবল যদি উভয় স্ট্রি এর ক্যারেকটারগুলো একই হয় এবং একই অর্ডারে থাকে। এক্ষেত্রে মনে রাখতে হবে বড় হাতের অক্ষর আর ছোট হাতের অক্ষরের ইউনিকোড আলাদা হওয়ায় ক্যারেকটার হিসেবেও এরা আলাদা হবে।

```
let quotation = "We're a lot alike, you and I."
let sameQuotation = "We're a lot alike, you and I."
if quotation == sameQuotation {
    println("These two strings are considered equal")
}
// prints "These two strings are considered equal"
```

স্ট্রিং টিতে কোন পার্টিকুলার (particular) স্ট্রিং প্রিফিক্স বা সাফিক্স আছে কিনা তা চেক করার জন্য রয়েছে hasPrefix বা hasSuffix মেথড। উভয় মেথড স্ট্রিং টাইপ আর্গুমেন্ট নিয়ে বুলিয়ান ভ্যালু রিটার্ন করে। আর্গুমেন্টের প্রত্যেকটি ক্যারেকটার মূল স্ট্রিং এ একই অর্ডারে আছে কিনা সেটাই চেক করে মেথড দুটি।

```
let comment = "Swift is a awesome language."
if comment.hasPrefix("Swift"){
    // String comment contains a "Swift" as prefix
}

if comment.hasSuffix("language."){
    // String comment contains a "language." as Suffix
}
```

উপরের উদাহরণটিতে comment একটি স্ট্রিং টাইপ কনস্ট্যান্ট নেওয়া হয়েছে। hasPrefix("Swift") মেথড টি মূলত comment স্ট্রিং এর প্রথমে "Swift" স্ট্রিং টি আছে কিনা। যেহেতু comment স্ট্রিং টিতে প্রিফিক্স হিসেবে "Swift" আছে তাই মেথডটি TRUE রিটার্ন করে। আবার hasSuffix("language.") মেথডটি এটাই চেক করছে যে comment স্ট্রিং টি "language." দিয়ে শেষ হয়েছে কিনা। যেহেতু স্ট্রিং টির শেষ "language." দিয়ে হয়েছে তাই মেথড টি TRUE রিটার্ন করে।

কেস(**Case**) কনভারসনঃ স্ট্রিং ম্যানিপুলেশনের একটি কমন টাস্ক হল কেস কনভারসন। অর্থাৎ ছোট হাতের অক্ষর থেকে বড় হাতের অক্ষর বা বড় হাতের অক্ষর থেকে ছোট হাতের অক্ষরে রূপান্তর করা। সুইফট এই কাজ যথেষ্ট সহজেই করা যায়।

```
let normal = "Would you mind giving me a glass of Water?"
let uppercase = normal.uppercaseString
// uppercase is equal to "WOULD YOU MIND GIVING ME A GLASS OF WATER?"
let lowercase = normal.lowercaseString
// lowercase is equal to "would you mind giving me a glass of water?"
```

পরিসমাপ্তিঃ নিশ্চয়ই খেয়াল করেছেন যে আমরা বার বার ইউনিকোড শব্দটি উচ্চারণ করছি। স্ট্রিং ও ক্যারেকটার হ্যান্ডলিং এর জন্য সুইফটের রয়েছে কিছু বিশেষ মেথড যা দিয়ে সহজেই স্ট্রিং ও ক্যারেকটারের ইউনিকোড রিপ্রেজেন্টেশন ম্যানিপুলেশন করা যায়। এসম্পর্কে বিস্তারিত আমাদের কাণ্ডজে বইতে পাওয়া যাবে।

বইয়ের আপডেট পেতে চোখ রাখুন আমাদের ফ্যান পেজে

পরের চাপ্টারঃ পরের চাপ্টারে Collections তথা Arrays ও Dictionaries নিয়ে বিস্তারিত থাকবে। এছাড়াও থাকবে Enumerations ও Closures নিয়ে অল্প বিস্তারিত আলোচনা।

Originally Posted [Here](#)

কালেকশনস (অ্যারে ও ডিকশনারী), ইনুমারেশন ও ক্লোজার

ভূমিকাঃ অ্যাপলের নতুন প্রোগ্রামিং ল্যান্ডস্কেপে সুইফট নিয়ে লেখা আমাদের তৃতীয় সেকশনের প্রথম অধ্যায়ে সুইফট সম্পর্কে পরিচিতি মূলক আলোচনা হয়েছে। আমরা সুইফট ল্যান্ডস্কেপের বেসিক সিনট্যাক্স, ভ্যারিয়েবল ও কনস্ট্যান্ট ডিক্লেয়ার করা, বিভিন্ন ধরনের অপারেটর সম্পর্কে জেনেছি। দ্বিতীয় অধ্যায়ে স্ট্রিং ও ক্যারেকটার নিয়ে বিস্তারিত আলোচনা হয়েছে। এই অধ্যায়ে অ্যারে, ডিকশনারী ইত্যাদি কালেকশনস (Collections) ও অল্পবিস্তর ইনুমারেশন ও ক্লোজার নিয়ে আলোচনা হবে।

কালেকশন টাইপঃ ধরুন, আপনি একজন রেস্টুরেন্টের মালিক। আপনি আপনার রেগুলার কাস্টমারদের একটি তালিকা করতে চাচ্ছেন সুইফট ল্যান্ডস্কেপে। একটু সহজ করার জন্য ধরে নিচ্ছি যে আপনি শুধু কাস্টমারদের নামের তালিকা করবেন। ভাবুন তো কিভাবে করবেন। যদি আপনার ১০০ জন কাস্টমার থাকে তাহলে ১০০ টা ভ্যারিয়েবল ডিক্লেয়ার করবেন। প্রত্যেকটিতে একজন করে কাস্টমারের নাম স্টোর করবেন।

```
var 1stCust = "First Customer's Name"
var 2ndCust = "2nd Customer's Name"
var 3rdCust = "3rd Customer's Name"
.
.
var 100thCust = "100th Customer's Name"
```

আচ্ছা, এবার বলুন তো আপনি যদি এই লিস্টটা দেখতে চান তাহলে কি করবেন?

```
println(1stCust)
println(2ndCust)
.
.
println(100thCust)
```

এভাবে করার সমস্যাটা হলঃ

- আপনি নিশ্চয়ই জানেন না যে আপনার কাস্টমার সংখ্যা কত হতে পারে।
- এভাবে ধরে ধরে এক-একটি ভ্যারিয়েবলে ডাটা স্টোর করতে হবে। আবার এক-একটি ধরে ডাটা ডিসপ্লে করতে হবে।
- কোন রকমের সার্চিং টেকনিক (Searching Technique) অ্যাপ্লাই করা যাবে না। অর্থাৎ কোন নির্দিষ্ট কাস্টমারের নাম খুঁজে বের করা যাবে না।
- কোন রকমের সর্টিং টেকনিক (Sorting Technique) অ্যাপ্লাই করা যাবে না। অর্থাৎ লিস্টটিকে কোন নির্দিষ্ট অর্ডারে সাজানো যাবে না।

এসব সমস্যার সমাধানের জন্য সুইফটে রয়েছে দুই ধরনের কালেকশন টাইপ (Collection Type) যথাঃ অ্যারে (Arrays) ও ডিকশনারী (Dictionaries)।

অ্যারে ও ডিকশনারী (**Arrays and Dictionaries**) ঃ একই ডাটাতাইপের অনেকগুলো ভ্যারিয়েবলের কালেকশনকেই অ্যারে অথবা ডিকশনারী বলা হয়। অ্যারে ও ডিকশনারী উভয়ই একটি নির্দিষ্ট ডাটাতাইপের এক বা একাধিক ডাটা সংরক্ষণ করতে পারে। অ্যারেতে ভ্যালুগুলো সবসময় একটি অর্ডারে সাজানো থাকে। 0 থেকে শুরু করে 1, 2, 3, ..., n-1 পর্যন্ত অর্ডারে ডাটাগুলো থাকে। এই 0 থেকে n-1 কে অ্যারের ইনডেক্স বলা হয়। এখানে n মানে হল অ্যারের সাইজ বা অ্যারেতে থাকা ভ্যালুর সংখ্যা। এই ইনডেক্স ব্যবহার করেই অ্যারের ভ্যালু গুলো ম্যানিপুলেট করা হয়। অন্যদিকে ডিকশনারীতে ভ্যালুগুলো কোন নির্দিষ্ট অর্ডারে সাজানো থাকে না। এখানে ইনডেক্সকে বলা হয় কি (Key)। key হল নির্দিষ্ট ইউনিক আইডেন্টিফায়ার যা দিয়ে ডিকশনারীর ভ্যালুগুলোকে সংরক্ষণ বা এক্সেস করা যায়। যেমন ঃ

```
restaurantArr[0] = 10; // Array
restaurantArr[1] = 15; // Array
restaurantDict["numberOfItem"] = 10 // Dictionary
```

অ্যারে (**Array**) একটি অ্যারেতে একই ডাটাতাইপের একাধিক ভ্যালু থাকতে পারে। 0 থেকে শুরু করে ১, ২, ৩, ... অর্ডারে সাজানো থাকে ভ্যালুগুলো। আবার একই ভ্যালু একাধিক বারও থাকতে পারে। অবজেকটিভ-সি এর NSArray এবং NSMutableArray তে যেকোন অবজেক্ট ভ্যালু হিসেবে রাখা যায়। এমনকি একটি অ্যারে তে বিভিন্ন টাইপের ভ্যালু বা অবজেক্ট থাকতে পারে। কিন্তু সুইফট -এ অ্যারেতে একটি নির্দিষ্ট টাইপের ডাটা রাখা যাবে। উদাহরণস্বরূপ Int টাইপের অ্যারেতে শুধু Int টাইপের ডাটাই রাখা যাবে। অন্য কিছু না।

অ্যারে টাইপ ভ্যারিয়েবল ডিক্লেয়ার করাঃ বিভিন্নভাবে অ্যারে টাইপ ভ্যারিয়েবল ডিক্লেয়ার বা ইনিশিয়ালাইজ করা যায়। নিচে দুই ধরনের ডিক্লেয়ারেশন এর উদাহরণ দেওয়া হলঃ

```
ArrayvariableA
var variableB: Int[] = [1, 2, 4]
```

উপরের কোডটিতে variableA এবং variableB নামের দুটি অ্যারে ডিক্লেয়ার করা হয়েছে। দুই ভাবেই অ্যারে ডিক্লেয়ার করা যায়। প্রথমটিকে ফুল ফর্ম আর দ্বিতীয়টিকে শর্টহ্যান্ড সিনট্যাক্স বলা হয়। শর্টহ্যান্ড সিনট্যাক্সই সবচেয়ে বেশী ব্যবহার করা হয়। এক্ষেত্রে var অথবা let লিখে তারপর অ্যারের নাম লিখতে হয়। এরপর কোলন (:) দিয়ে ডাটাতাইপ লিখে তারপর বন্ধনী ([]) চিহ্ন দিতে হয়।

অ্যারেতে সরাসরি [value1, value2,.....,valueN] অ্যাসাইন করে অ্যারে ইনিশিয়ালাইজ করা যায়। যেমনঃ

```
var variableInt: Int[] = [12, 12, 34]
//variableInt is initialised with 12,12 and 34

var variableString = ["Egg", "Fruits"]
// variableString is initialised with Egg and Fruits with these two string
```

সুইফট এর টাইপ ইনফারেন্স (Type Inference) সুবিধার জন্য অ্যারে ডিক্লেয়ার বা ইনিশিয়ালাইজেশনের সময় অ্যারের টাইপ না লিখলেও অ্যাসাইন করা ভ্যালুর টাইপই হবে অ্যারে টাইপ। তাই variableString একটি String টাইপ অ্যারে কারন দুটি String টাইপের ভ্যালু দিয়ে অ্যারেটি ইনিশিয়ালাইজ করা হয়েছে।

অ্যারে ম্যানিপুলেশন (এক্সেস ও মডিফিকেশন) অ্যারের ড্যালাগুলো পড়া (Access) বা আপডেট (Modification) করার জন্য সুইফট ল্যাঙ্গুয়েজের অ্যারে টাইপ ডারিভেটের জন্য কিছু প্রোপার্টি, মেথড আছে। এসব প্রোপার্টি বা মেথড দিয়ে অথবা সাবস্ক্রিপ্ট দিয়ে অ্যারে এক্সেস বা মডিফাই করা যায়। নিচে কিছু উদাহরণ দিয়ে বিস্তারিত বর্ণনা করা হলঃ কোন অ্যারেতে কতগুলো ড্যালা আছে তা জানা যাবে রিড-অনলি প্রোপার্টি count দিয়ে।

```
var menuItems = ["Eggs", "Potatoes", "Chilis"]
println("menuItems array has \(menuItems.count) items.") // menuItems array has 3 items.
```

প্রথমে menuItems অ্যারেটি ৩ টি স্ট্রিং টাইপ ড্যালা দিয়ে ইনিশিয়ালাইজ করা হয়েছে। তারপর menuItems.count প্রোপার্টিতে অ্যারেতে কয়টি ড্যালা আছে তা রিটার্ন করে। তাই menuItems.count ৩ রিটার্ন করবে।

বুলিয়ান isEmpty প্রোপার্টি দিয়ে কোন অ্যারে ফাঁকা কিনা তা চেক করা যায়। অর্থাৎ যদি কোন অ্যারের count ০ হয় বা অ্যারেতে কোন ইলিমেন্ট না থাকে তাহলে isEmpty প্রোপার্টি YES (true) রিটার্ন করে।

```
if menuItems.isEmpty {
    println("The menu list is empty.")
} else {
    println("The menu list is not empty.")
}
// prints "The menu list is not empty."
```

যেহেতু menuItems অ্যারেতে ৩টি স্ট্রিং ইলিমেন্ট আছে তাই menuItems.isEmpty প্রোপার্টি NO রিটার্ন করবে।

অ্যারের append মেথড ও += অপারেটর দিয়ে অ্যারেতে নতুন ড্যালা যোগ করা যায়। আবার সরাসরি একটি নতুন অ্যারে কে += অপারেটর দিয়ে অ্যারের শেষে যোগ করা যায়। নিচে উদাহরণ দেওয়া হলঃ

```
menuItems.append("Oils")
//Oils is appended on the menuItems array. Now menuItems has 4 strings.

menuItems += "Flour"
// Flour is appended on the menuItems array. Now menuItems has 5 strings

menuItems += ["Baking powder", "Butter", "Chocolate" ]
//An array of 3 new items is appended on the menuItems array. Now menuItems has 8 strings
```

আগেই বলেছি অ্যারেতে ড্যালাগুলো ০, ১, ২, ... অর্ডারে সাজানো থাকে। যেমন অ্যারের ৫ নম্বর ড্যালা তথা ৫-১ বা ৪ তম ইনডেক্স এর ড্যালা এক্সেস করার জন্য সাবস্ক্রিপ্ট সিনট্যাক্স ব্যবহার করে নিচের মত স্কয়ার ব্র্যাকেটের মধ্যে ইনডেক্স পাঠাতে হয়।

```
var 5thItem = menuItems[4]
```


এক্ষেত্রে অবশ্যই খেয়াল রাখতে হবে যাতে পাঠানো ইনডেক্সটি অ্যারেতে থাকা ড্যালাুর সংখ্যার সর্বোচ্চ ইনডেক্সের বেশী না হয়। ধরুন অ্যারেতে ০ থেকে ৪ ইনডেক্সে ৫ টি ড্যালাু রয়েছে। কিন্তু আপনি ৫ নম্বর ইনডেক্স তথা ৬ষ্ঠ ড্যালাু চাচ্ছেন। সেক্ষেত্রে "Array out of bound" টাইপের এরর হতে পারে।

একই ভাবে সাবস্ক্রিপ্ট সিনট্যাক্স দিয়ে কোন নির্দিষ্ট ইনডেক্স বা কোন রেঞ্জের মধ্যকার সকল ইনডেক্সের ড্যালাু আপডেট বা মডিফিকেশন করা যায়।

```
menuItems[4] = "Onion" // "Flour" is replaced by "Onion"
menuItems[1..3] = ["Bananas", "Apples"]
// Potatoes and Chilis are replaced by Bananas and Apples
```

এখানে, দ্বিতীয় উদাহরণটিতে 1..3 মানে 1 থেকে 3 রেঞ্জের (শেষ ইনডেক্স, এক্ষেত্রে ৩য় ইনডেক্স কমিডার হয় না) সকল ইনডেক্সের ড্যালাুকে একটি অ্যারে দিয়ে রিপ্লেস করা হচ্ছে যাতে দুটি ড্যালাু রয়েছে। অর্থাৎ সাবস্ক্রিপ্ট সিনট্যাক্স ব্যবহার করে যেকোন সংখ্যক ইনডেক্সের ড্যালাু অন্য যেকোন সংখ্যক ড্যালাু দিয়ে রিপ্লেস করা যাবে। উল্লেখযোগ্য কথা হল সাবস্ক্রিপ্ট সিনট্যাক্স (স্কয়ার ব্র্যাকেটের ভিতর অ্যারে ইনডেক্স পাঠানো) ব্যবহার করে অ্যারের ড্যালাু রিড করা বা নতুন ড্যালাু দিয়ে রিপ্লেস করা যায় ঠিকই কিন্তু নতুন কোন ড্যালাু অ্যারের শেষে ইনসার্ট (Add/ Append/ Insert) করা যায় না।

```
var recipes = ["Chicken Resala", "Mutton Curry"]
recipes[2] = "another recipe" // Throws run time error because index 2 doesn't have val
println(recipes[2]) // Throws run time error because index 2 doesn't have value
```

recipes অ্যারেতে ০, ১ ইনডেক্সে মোট দুটি ড্যালাু রয়েছে। যদি ইনডেক্স ২ এর ড্যালাু এক্সেস করার বা নতুন ড্যালাু সেট করার চেষ্টা করা হয় তাহলে রান টাইম এরর হয়। অর্থাৎ সাকসেসফুল কম্পাইলেশনের পর কোড রান করলে যখনই মেশিন recipes অ্যারের ২ নম্বর ইনডেক্সের ড্যালাু এক্সেস করতে চাইবে তখনই এরর হবে কারন এই অ্যারেতে ২ নম্বর ইনডেক্স নেই।

কোন নির্দিষ্ট ইনডেক্সে নতুন ড্যালাু ইনসার্ট করার জন্য রয়েছে অ্যারের Insert মেথড। Insert মেথডে দুটি আরগুমেন্ট পাঠানো হয়। প্রথমে নতুন ড্যালাু ও তারপর যে ইনডেক্সে ইনসার্ট করতে হবে তার ড্যালাু।

```
menuItems.insert("Maple Syrup", atIndex: 2)
```

menuItems অ্যারের ২ নম্বর ইনডেক্সে Maple Syrup ইনসার্ট হয়ে গেছে। পরবর্তী সকল ড্যালাুর ইনডেক্স ১ করে বেড়ে গেছে। অর্থাৎ এই ইনসার্ট অপারেশন চালানোর পূর্বে যে ড্যালাুটি ইনডেক্স ২ তে ছিল এখন সেটি ইনডেক্স ৩ এ আছে।

নিচের মত করে removeAtIndex() মেথড দিয়ে অ্যারের যেকোন ইনডেক্সের ড্যালাু রিমুভ করা যায়। সেক্ষেত্রে রিমুভ করার পর ওই ইনডেক্সের পরবর্তী সকল ড্যালাুর ইনডেক্স ১ করে কমে যাবে।

```
menuItems.removeAtIndex(2)
```

ইনডেক্স ২ এর ভ্যালু রিমুভ হয়ে পরবর্তীতে থাকা সকল ভ্যালুর ইনডেক্স ১ করে কমে যাবে।

`removeLast()` মেথড দিয়ে সহজেই যেকোন অ্যারের শেষ ভ্যালুটি রিমুভ করা যায়। ফলে অ্যারের `count` ও ১ কমে যায়।

```
menuItems.removeLast() // removes the last element
```

`for-in` লুপ ব্যবহার করে অ্যারের সবগুলো ইনডেক্স ইটরেট করা যায়। নিচের উদাহরণটি দেখা যাকঃ

```
for item in menuItems
{
    println(item)
}
```

অ্যারে ইনিশিয়লাইজেশনঃ শুরুতে কোন ভ্যালু ছাড়াই ফাঁকা অ্যারে ডিক্লেয়ার করার জন্য অ্যারের ডাটাটাইপ লিখে তারপর স্কয়ার ব্র্যাকেট লিখতে হয় নিচের মত করে।

```
var someInts = Int[]()
println("someInts is of type Int[] with \$(someInts.count) items.")
```

সুইফট এর `Array` তে রয়েছে এমন একটি ইনিশিয়লাইজার যা দিয়ে একটি নির্দিষ্ট সাইজের অ্যারে ডিক্লেয়ার করা যাবে এবং প্রতিটি ইনডেক্সে একটি ডিফল্ট ভ্যালু সেট করা যাবে। উদাহরণস্বরূপ ধরুন আপনি চাচ্ছেন এমন একটি অ্যারে ডিক্লেয়ার করতে যার সাইজ (`count`) হবে ১০০ (ইনডেক্স ০ - ৯৯) এবং প্রতিটি ইনডেক্সের ডিফল্ট ভ্যালু হবে ০.০ (`repeatedValue`)।

```
var hundredDoubles = Double[(count: 100, repeatedValue: 0.0)]
// hundredDoubles is of type Double[], and equals [0.0, 0.0, 0.0, ...]
```

`hundredDoubles` অ্যারেটি ডিক্লেয়ার করার সময় ব্র্যাকেটের মধ্যে দুটি আর্গুমেন্ট পাঠানো হয়। প্রথমটি `count` যা অ্যারের সাইজ কত হবে তা ডিফাইন করে। দ্বিতীয়টি `repeatedValue` যা অ্যারের সকল ইনডেক্সের ডিফল্ট ভ্যালু কত হবে তা সেট করে। আবার `Double[]` লিখে অ্যারের ভ্যালুগুলোর ডাটাটাইপ কি হবে তাও বলে দেওয়া হয়েছে। সুইফট এর টাইপ ইনফারেন্স সুবিধার জন্য ডাটাটাইপ না বলে দিয়ে নিচের মত করে অ্যারে ইনিশিয়লাইজ করা যায়।

```
var anotherThreeDoubles = Array(count: 3, repeatedValue: 2.5)
// anotherThreeDoubles is inferred as Double[], and equals [2.5, 2.5, 2.5]
```

একথা বলাই বাহুল্য যে দুটি অ্যারে কে "+" অপারেটর দিয়ে যোগ করলে প্রথম অ্যারের শেষে দ্বিতীয় অ্যারেটি অ্যাপেন্ড হয়।

```
var sixDoubles = threeDoubles + anotherThreeDoubles
// sixDoubles is inferred as Double[], and equals [0.0, 0.0, 0.0, 2.5, 2.5, 2.5]
```

ডিকশনারী (Dictionaries) :: অ্যারের মতই ডিকশনারীতে একই টাইপের একাধিক ভ্যালু স্টোর করা যায়। প্রত্যেকটি ভ্যালুর জন্য একটি করে ইউনিক কি (key) বা আইডেন্টিফায়ার থাকে। অ্যারেতে ০ থেকে শুরু করে ১, ২, ৩, ... ইনডেক্স গুলোতে ভ্যালু গুলো থাকে কিন্তু ডিকশনারীতে ভ্যালুগুলো অ্যারের মত ইনডেক্সিং করে সাজানো থাকে না। ঠিক বাস্তব ডিকশনারীতে যেমন নির্দিষ্ট শব্দের জন্য নির্দিষ্ট সংজ্ঞা থাকে তেমনি সুইফট এর ডিকশনারীতে প্রত্যেকটি ভ্যালু নিজের ইউনিক আইডেন্টিফায়ার দিয়ে স্টোর করা থাকে। এই আইডেন্টিফায়ার বা ভ্যালু উভয়টি যেকোন টাইপের হতে পারে। তবে একটি ডিকশনারীর সকল ভ্যালুর টাইপ একই হবে এবং সকল আইডেন্টিফায়ারের টাইপ একই হবে।

অবজেক্টিভ-সি এর NSDictionary ও NSMutableDictionary তে একই ডিকশনারীতে বিভিন্ন টাইপের ভ্যালু বা আইডেন্টিফায়ার (key) থাকতে পারে। ফলে ডিকশনারীর টাইপ নির্দিষ্ট থাকে না। অন্যদিকে যেহেতু সুইফটের ডিকশনারীর সকল ইলিমেন্ট এর টাইপ একই হয় তাই ডিকশনারীর ভ্যালুর টাইপ সম্পর্কে নিশ্চিত থাকা যায়।

ডিকশনারী (Dictionaries) ডিক্লেয়ারেশন :: সুইফট ডিকশনারী টাইপ ডিক্লেয়ার করার জন্য Dictionary লিখতে হয়। এখানে KeyType হল সেই আইডেন্টিফায়ার বা কি (key) এর টাইপ এবং ValueType হল ডিকশনারীতে স্টোর করা ভ্যালুর টাইপ। যেমনঃ

```
var airports: Dictionary = ["TYO": "Tokyo", "DUB": "Dublin"]
```

ডিকশনারী ইনিশিয়ালাইজ করার জন্য অ্যারের মতই শর্টহ্যান্ড সিনট্যাক্স ব্যবহার করা হয়। অর্থাৎ ভ্যারিয়েবল ডিক্লেয়ার করার সময় স্কয়ার ব্র্যাকেটের ভিতর ভ্যালুগুলো লিখে দেওয়া। কিন্তু ডিকশনারীতে যেহেতু অ্যারের মত ইনডেক্সিং হয় না, তাই এখানে কি-ভ্যালু পেয়ার (key-value pairs) দিয়ে ডিকশনারী ইনিশিয়ালাইজ করতে হয়। উপরের উদাহরণটিতে ডিকশনারীর ইলিমেন্টগুলো কমা (,) দিয়ে লেখা হয়েছে। এবং প্রত্যেক ইলিমেন্ট এর কোলন (:) দিয়ে কি-ভ্যালু পেয়ার লেখা হয়েছে। এখানে কোলনের ডানদিকের অংশটি হল ভ্যালু ও বামদিকের অংশটি হল এই ভ্যালুর জন্য ইউনিক আইডেন্টিফায়ার বা কি (key)।

নোট :: আমরা বার বার বলছি যে ডিকশনারীর কি (key) গুলো অবশ্যই ইউনিক হতে হবে। তা না হলে একই কি(key) এর জন্য যদি একাধিক ভ্যালু স্টোর করা হয় তাহলে তা Ambiguity তৈরী করবে। এজন্যই সুইফটে কি(key) এর জন্য এমন ডাটাটাইপ ব্যবহার করতে হবে যেগুলোকে হ্যাশ করা যায়। String, Int, Double, Bool ইত্যাদি ডাটাটাইপ বাই-ডিফল্ট হ্যাশ করা যায়। তাই এসবগুলো ডাটাটাইপই ডিকশনারীর কি(key) হিসেবে ব্যবহার করা যাবে।

ডিকশনারী ম্যানিপুলেশন (এক্সেস ও মডিফিকেশন) :: ডিকশনারীর ভ্যালুগুলো পড়া (Access) বা আপডেট (Modification) করার জন্য সুইফট ল্যান্গুয়েজের ডিকশনারী টাইপ ভ্যারিয়েবলের জন্য আছে কিছু প্রোপার্টি, মেথড। এসব প্রোপার্টি বা মেথড দিয়ে অথবা সাবস্ক্রিপ্ট দিয়ে ডিকশনারী এক্সেস বা মডিফাই করা যায়। নিচে কিছু উদাহরণ দিয়ে বিস্তারিত বর্ণনা করা হলঃ

কোন ডিকশনারীতে কতগুলো ভ্যালু আছে তা জানা যাবে রিড-অনলি প্রোপার্টি count দিয়ে।

```
var airports: Dictionary = ["TYO": "Tokyo", "DUB": "Dublin"]
println("airports array has \(menuItems.count) items.") // airports dictionary has 2 item
```

প্রথমে airports ডিকশনারীটি 2 টি স্ট্রিং টাইপ ভ্যালু দিয়ে ইনিশিয়ালাইজ করা হয়েছে যাদের কি-ভ্যালু পেয়ার হল String : String । তারপর airports.count প্রোপার্টিতে ডিকশনারীতে কয়টি ভ্যালু আছে তা রিটার্ন করে । তাই airports.count 2 রিটার্ন করবে ।

সাবস্ক্রিপ্ট সিনট্যাক্স দিয়ে সহজেই ডিকশনারীতে নতুন ভ্যালু ইনসার্ট বা আগের কোন ভ্যালু আপডেট করা হয় । সাবস্ক্রিপ্ট এ ইনডেক্স হিসেবে কি (key) লিখে তাতে নতুন ভ্যালু অ্যাসাইন করলে প্রথমেই চেক করা হয় যে এই কি (key) দিয়ে কোন ভ্যালু ডিকশনারীতে আছে কিনা । যদি থাকে তাহলে নতুন ভ্যালু দিয়ে আপডেট হবে । আর যদি না থাকে তাহলে নতুন ইলিমেন্ট টি ডিকশনারীতে ইনসার্ট হবে ।

```
airports["LHR"] = "London" // new item with "LHR" : "London" key-pair is inserted
// the airports dictionary now contains 3 items

airports["LHR"] = "London Heathrow" // existing item of key "LHR" is updated by ""London
// the value for "LHR" has been changed to "London Heathrow"
```

সাবস্ক্রিপ্ট সিনট্যাক্স ব্যবহার না করে ডিকশনারীর updateValue(forKey:) মেথড দিয়ে কোন নির্দিষ্ট কি (key) এর ভ্যালু আপডেট করা যায় আবার নতুন কি-পেয়ার ভ্যালু ইনসার্ট করা যায় ।

```
if let oldValue = airports.updateValue("Dublin International", forKey: "DUB") {
    println("The old value for DUB was \(oldValue).")
}
// prints "The old value for DUB was Dublin."
```

উপরের কোডটিতে দেখা যাচ্ছে updateValue মেথডে দুটি আর্গুমেন্ট পাঠানো হচ্ছে । প্রথমটিতে ভ্যালু এবং দ্বিতীয়টিতে কি(key) । এখানে ভ্যালু হিসেবে "Dublin International" এবং forKey:"DUB" এ কি (key) হিসেবে "DUB" পাঠানো হচ্ছে । প্রথমে চেক করা হচ্ছে যে DUB কি দিয়ে কোন ইলিমেন্ট ডিকশনারীতে আছে কিনা । যেহেতু airports ডিকশনারীতে এই কি দিয়ে একটি ভ্যালু আছে তাই এই কি এর পূর্বের ভ্যালুটি নতুন পাঠানো Dublin International দিয়ে আপডেট হয়ে গেছে এবং পূর্বের ভ্যালুটি রিটার্ন করে যা oldValue ভ্যারিয়েবলে অ্যাসাইন হয় । যদি airports ডিকশনারীতে এই কি দিয়ে কোন ভ্যালু না থাকত, তাহলে নতুন ইলিমেন্ট হিসেবে "DUB" : "Dublin International" কি-পেয়ার ডিকশনারীতে যুক্ত হত ।

সাবস্ক্রিপ্ট সিনট্যাক্সে স্কয়ার ব্র্যাকেটের মধ্যে কি(key) লিখে ওই কি এর ভ্যালু রিড বা এক্সেস করা যায় ।

```
var airport = airports["DUB"]
println(airport) // Dublin International
```

কোন নির্দিষ্ট কি এর ভ্যালু রিমুভ করতে চাইলে তার সাবস্ক্রিপ্ট এ nil অ্যাসাইন করলেই তা ডিকশনারী থেকে রিমুভ হয়ে যাবে ।

```
airports["DHK"] = "Shahjalal International Airport"
// airports contains 3 items right now.

airports["DHK"] = nil
// Value for "DHK" key is removed from airports and this dictionary contains two items
```

আবার সাবস্ক্রিপ্ট ব্যবহার না করে `removeValueForKey()` মেথড দিয়ে কোন নির্দিষ্ট কি-ভ্যালু পেমার ডিকশনারী থেকে রিমুভ করা যায়। এক্ষেত্রে মেথডটি রিমুভ হওয়া ভ্যালুটি রিটার্ন করে।

```
if let removedValue = airports.removeValueForKey("DUB") {
    println("The removed airport's name is \(removedValue).")
} else {
    println("The airports dictionary does not contain a value for DUB.")
}
// prints "The removed airport's name is Dublin International."
```

অ্যারের মতই `for-in` লুপ দিয়ে ডিকশনারীতে থাকা সকল ভ্যালুতে ইটরেট করা যায়। ডিকশনারীতে কি-ভ্যালু পেমার হিসেবে ভ্যালু থাকে তাই এখানে অ্যারের মতই একই সাথে কি এবং ভ্যালু এক্সেস করা যায়।

```
for (airportCode, airportName) in airports {
    println("\(airportCode): \(airportName)")
}
// TYO: Tokyo
// LHR: London Heathrow
```

আবার চাইলে শুধু keys অথবা শুধু values প্রোপার্টি ইটরেট করে শুধু কি বা শুধু ভ্যালুগুলো এক্সেস করা যায়।

```
for airportCode in airports.keys {
    println("Airport code: \(airportCode)")
}
// Airport code: TYO
// Airport code: LHR

for airportName in airports.values {
    println("Airport name: \(airportName)")
}
// Airport name: Tokyo
// Airport name: London Heathrow
```

একটি ফাঁকা ডিকশনারী (**Empty Dictionary**) ইনিশিয়ালাইজ করা `Dictionary()` ইনিশিয়ালাইজার সিনট্যাক্স ব্যবহার করে অ্যারের মতই ফাঁকা ডিকশনারী ইনিশিয়ালাইজ করা যায়।

```
var namesOfIntegers = Dictionary()
// namesOfIntegers is an empty Dictionary
```

উদাহরণটিতে `nameofIntegers` নামের একটি ডিকশনারী ইনিশিয়ালাইজ করা হয়েছে যার কি গুলো `Integer` টাইপ, কিন্তু ভ্যালুগুলো হবে `String` টাইপ।

কোন ডিকশনারী ইনিশিয়ালাইজ করার সময় যদি টাইপ বলে দেওয়া হয় অথবা কোন ভ্যালু ইনিশিয়ালাইজ করার ফলে টাইপ ইনফারেন্স ঘটে তাহলে ডিকশনারী টি ফাঁকা করলেও টাইপ পরিবর্তন হয় না। এই বৈশিষ্ট্য অ্যারের ক্ষেত্রেও প্রযোজ্য।

```
namesOfIntegers[16] = "sixteen"
// namesOfIntegers now contains 1 key-value pair
namesOfIntegers = [:]
// namesOfIntegers is once again an empty dictionary of type Int, String
```

কোন ডিকশনারী তে `[:]` এভাবে স্বয়ং ব্র্যাকেটের ভিতর শুধু কোলন (`:`) দিয়ে অ্যাসাইন করলে তা ফাকা ডিকশনারীতে পরিনত হয়।

ইনুমারেশন ও ক্লোজার (**Enumerations and Closures**) : ইনুমারেশন দিয়ে একই ধরনের একগুচ্ছ ভ্যালু টাইপকে প্রকাশ করা হয়। আপনি যদি `সি/সি++` এর সাথে পরিচিত থাকেন তাহলে নিশ্চয়ই জানেন যে, `সি` তে একটি এনাম (`enum`) টাইপ গ্রুপের সকল ভ্যারিয়েবলকে ইনটিজার (`Integer`) ভ্যালু দিয়ে অ্যাসাইন করা হয়। সুইফ্ট এ ইনুমারেশন অনেক বেশী ফ্লেক্সিবল এবং `সি` এর মত গ্রুপের ভ্যালুগুলো শুধু ইনটিজার ভ্যালু হয় না। যেকোন ডাটাতাইপ যেমন `String`, `Character`, `Integer` বা `Float` টাইপ হতে পারে।

```
enum CompassPoint {
    case North
    case South
    case East
    case West
}
```

সব ল্যাম্বুয়েজেই `enum` কিওয়ার্ড দিয়ে ইনুমারেশন টাইপ ডিক্লেয়ার করা হয়। উপরের উদাহরণটিতে `CompassPoint` নামে একটি `enum` টাইপ ডিক্লেয়ার করা হয়েছে যার চারটি সদস্য (`member`) যথাক্রমে `North`, `South`, `East` এবং `West` রয়েছে। `সি/সি++` বা অন্য যেকোন ল্যাম্বুয়েজে `enum` টাইপের প্রত্যেকটি সদস্য (`member`) এ ০ থেকে ১, ২, ৩, ... `integer` ভ্যালু অ্যাসাইন হয়। কিন্তু সুইফ্ট এর `enum` টাইপ এর `member` দেব ভ্যালুতে এরকম কোন ডিফল্ট ভ্যালু অ্যাসাইন করা হয় না।

কোন নির্দিষ্ট ফাংশনালিটি সহ একটি কোড ব্লকে ক্লোজার (`Closure`) বলা হয় (`{ }`)। সুইফ্টের ক্লোজার ঠিক অবজেকটিভ-সি এর ব্লক (`Block`) এর মতই কাজ করে। নিচে ক্লোজারের একটি জেনারেল ফর্ম লেখা হল।

```
{ (param: Type) -> ReturnType in
    expression_using_params
}
```

উপরের `"->"` চিহ্নটি আর্গুমেন্ট ও রিটার্ন টাইপকে আলাদা করে এবং `"in"` ক্লোজার এর হেডার থেকে ক্লোজার এর বডি আলাদা করে। নিচে একটি উদাহরণ,

```
var numbers = [1, 2, 3, 4, 5]
numbers.map({
  (number: Int) -> Int in
  let result = 3 * number
  return result
})
```

যদি টাইপ জানা থাকে (যেমন উপরে), তাহলে নিচের মত করেও এই ক্লোজারটি ব্যবহার করা যেতে পারে,

```
var numbers = [1, 2, 6]
numbers = numbers.map({ number in 3 * number })
println(numbers) // [3, 6, 18]
```

পরিসমাপ্তিঃ এ অধ্যায়ে কালেকশন টাইপ নিয়ে বিস্তারিত আলোচনা হয়েছে। আমরা দেখেছি কিভাবে অ্যারে ও ডিকশনারী ইনিশিয়ালাইজ করা ও ম্যানিপুলেট করতে হয়। এছাড়া জেনেছি অ্যারে ও ডিকশনারীর মধ্যকার পার্থক্য ও সুবিধা-অসুবিধা। এছাড়া ইনুমারেশন ও ক্লোজার সম্পর্কে নামমাত্র আলোচনা হয়েছে। এ অধ্যায় সম্পর্কিত আরও বিস্তারিত থাকবে আমাদের কাণ্ডজে বইয়ে।

বইয়ের আপডেট পেতে চোখ রাখুন আমাদের [ফ্যান পেজে](#)

Originally Posted [Here](#)

এই সেকশনে থাকছে

- গেমস বনাম অ্যাপ্লিকেশন, গেমস বনাম গেমস (গেমসের প্রকারভেদ)
- অতিরিক্ত প্রয়োজনীয় টুলস এবং তাদের পরিচিতি, ফ্রেমওয়ার্ক নির্বাচন (স্প্রাইটকিট, কোকোস-২D, ...)
- স্থানাংক ব্যবস্থা, দেয়ালের প্রথম ইট
- ব্যাসিক কিছু ধারণা: স্প্রাইট(ছবি), অ্যানিমেশন, ফন্ট, সাউন্ড(শব্দ)
- গেমঃ মহাকাশ-যুদ্ধ
 - একটি স্পেসশিপ তৈরি
 - গুলি ছোঁড়া
 - প্রতিপক্ষ স্পেসশিপ(গুলো)
 - আক্রমণ
- গেম-সেন্টার
 - লিডার-বোর্ড (সেরাদের-তালিকা)
 - অ্যাচিভমেন্টস (অর্জন)
 - চ্যালেঞ্জ

এই সেকশনে থাকছে

- অ্যাপল ডেভেলপার আইডি খোলা
- আসল ডিভাইসে অ্যাপ রান এর পদ্ধতি
- আরও চ্যাপ্টার আসছে ...

এই সেকশনে থাকছে

- অবজেক্টিভ-সি (Objective-C) নাকি সুইফট (Swift)?
- আরও চ্যাপ্টার আসছে ...

অবজেক্টিভ-সি (Objective-C) নাকি সুইফট (Swift)?

ভূমিকাঃ আপনি যদি Apple এর WWDC (Worldwide Developer's Conference) সম্পর্কে মোটা মুটি অবগত থাকেন অথবা ইনফরমেশন টেকনোলজি সম্পর্কিত আন্তর্জাতিক খবর গুলো খেয়াল করে থাকেন, তাহলে জেনে থাকবেন যে Apple তাদের WWDC 2014 ইভেন্টে সবচেয়ে চমকপ্রদ যে আবিষ্কারটির ঘোষণা দিয়েছে তা হচ্ছে তাদের তৈরি সম্পূর্ণ নতুন একটি প্রোগ্রামিং ল্যাঙ্গুয়েজের খবর। যার নাম Swift. তারা চায় তাদের ভবিষ্যৎ iOS এবং OSX অ্যাপ্লিকেশন গুলো এই ল্যাঙ্গুয়েজ দিয়েই ডেভেলপ করা হোক যাতে করে এই প্ল্যাটফর্মের অ্যাপ গুলোর পারফরমেন্স আরও ভালো হয়। এটাকে তারা বলছে, দ্রুতগতি সম্পন্ন, আধুনিক, নিরাপদ ও ইন্টারঅ্যাক্টিভ একটি ল্যাঙ্গুয়েজ। অন্যান্য ল্যাঙ্গুয়েজের মত অনেক অনেক জনপ্রিয় ফিচার এই ল্যাঙ্গুয়েজে যুক্ত আছে। এর ডিজাইন এমন ভাবে করা হয়েছে যাতে সিনট্যাক্স আরও সহজ হয় এবং iOS ও OSX ডেভেলপমেন্ট শুরু করতে নতুনদের বাধা আরও কম হয়। এমনকি আসছে সেপ্টেম্বর, ২০১৪ তে যে Xcode 6 লঞ্চ হতে যাচ্ছে তার সঙ্গে Playground নামের একটি ফিচার থাকছে যার মাধ্যমে বিভিন্ন কোড, প্রোগ্রামিং লজিক এবং ক্যালকুলেশনের লাইভ প্রিভিউ দেখা যাবে পুরো প্রোগ্রাম রান না করেই। অর্থাৎ Apple বরাবরই ডেভেলপার ফ্রেন্ডলি একটা ডেভেলপমেন্ট প্ল্যাটফর্ম দেয়ার ব্যাপারে সবসময় গুরুত্ব দিয়েছে যারই বহিঃপ্রকাশ হিসেবে Swift এর জন্ম বলতে পারেন। অতএব, ভয় না পেয়ে এর কাছ থাকে ভালো কিছাই আশা করতে পারেন নতুন এবং পুরনো iOS এবং OSX ডেভেলপারেরা।

আমি এই প্ল্যাটফর্মে নতুন, আমার কি এখন অবজেক্টিভ-সি অথবা সুইফট নাকি দুটো ল্যাঙ্গুয়েজ-ই শেখা উচিত? প্রথমত, সুইফট (Swift) একটি নতুন প্রোগ্রামিং ল্যাঙ্গুয়েজ, আর তাই এটাতে আরও নতুন নতুন ফিচার যুক্ত হওয়া থেকে শুরু করে বিভিন্ন বাগ ফিক্সিং চলতেই থাকবে সামনের অল্পত এক দুই বছর। আর তাই Apple এটার ব্যাপারে প্রচার চালিয়ে যাবে ঠিকই কিন্তু আপনাকে বাধ্য করবে না iOS এর অ্যাপ শুধুমাত্র Swift এ করার জন্য। আর অন্যদিকে অবজেক্টিভ-সি রাতারাতি বন্ধও হয়ে যাবে না। দ্বিতীয়ত, ইতোমধ্যে Apple অ্যাপ স্টোরে ১০ লাখেরও বেশি অ্যাপ্লিকেশন আছে যেগুলো অবজেক্টিভ-সি তে করা এবং ওয়েবে কয়েক লাখ জনপ্রিয় লাইব্রেরি, ফ্রেমওয়ার্ক ওপেন সোর্স টুলস ও প্রজেক্ট আছে যেগুলোও অবজেক্টিভ-সি তে ডেভেলপ করা। আর তাই এগুলোর এনহ্যান্সমেন্ট, বাগ ফিক্সিং এবং আপগ্রেড চলবে আরও অনেক দিন আর তার জন্য অবশ্যই অবজেক্টিভ-সি তে অভিজ্ঞ ডেভেলপার বা প্রোগ্রামারের প্রয়োজন থাকছেই। তৃতীয়ত, Swift এবং iOS 7,8 সাথে Xcode 6 এমন ভাবে প্রস্তুত আছে যে আপনি একটি প্রোজেক্টে একি সাথে অবজেক্টিভ-সি এবং সুইফট ল্যাঙ্গুয়েজ ব্যবহার করতে পারেন কোন রকম বাড়তি ঝামেলা ছাড়াই। আর এই যুগপৎ বিদ্যমানতা এটাই প্রমাণ করে যে, সুইফট একবারেই অবজেক্টিভ-সি এর জায়গা দখল করে নিচ্ছে না। আরও দেখতে পারেন [এখানে](#)।

আর তাই, যদি আপনি কোন iOS ডেভেলপার কোম্পানিতে জয়েন করতে চান অথবা নিজে থেকেই এই মার্কেটে অ্যাপ লঞ্চ করতে চান আপনাকে দুটো ল্যাঙ্গুয়েজেই সম্যক ধারণা নেয়া খুব গুরুত্বপূর্ণ।

আমি বেশকিছু দিন ধরেই অবজেক্টিভ-সি তে অ্যাপ ডেভেলপমেন্ট এর কাজ করে আসছি কিন্তু এখন কি আমি একজন কেবলই নতুন শিক্ষানবিস? একদম না। চিন্তা করে দেখুন, আপনি ইতোমধ্যে Xcode, Cocoa এবং Cocoa Touch এর বিভিন্ন API এবং অবজেক্টিভ-সি তে অভিজ্ঞতা অর্জন করেছেন যার মাধ্যমে চলছে কয়েক লাখ অ্যাপ - তার তুলনায় Swift শেখা কিছাই না। বরং আপনি আপনার অভিজ্ঞতার ভাণ্ডারে নতুন একটি জিনিষ যুক্ত করতে যাচ্ছেন মাত্র। অন্যদের থেকে তার মানে আপনি সিংহ ভাগ এগিয়ে থাকছেনই সব সময়।

সুইফ্ট দিয়ে ডেভেলপমেন্টের সুবিধা কি? Apple এর মতে এটা ৩০ বছর বয়সী Objective-C এর চেয়ে অনেকটাই আধুনিক। আর তাই এতে প্রোগ্রামারদের অনেক প্রিয় কিছু ফিচার যেমন namespaces, optionals, tuples, generics, type inference ইত্যাদি থাকছে যা অবশ্যই সফটওয়্যার ডেভেলপমেন্টকে আরও বেশি যুগোপযোগী আর গুনমান সম্পন্ন করবে। অন্যদিকে এই ল্যান্ডস্কেপের অবজেক্ট সর্টিং, এক্সিকিউশন সহ আরও কিছু বিষয়ে টাইম কমপ্লেক্সিটি অনেক কম।

কোথায় শেখা শুরু করবো? সবসময় নতুন কিছু শুরু করতে বা ওই বিষয়ে জানতে সেটার অফিসিয়াল সোর্স থেকেই দেখে নেয়া উচিত। যেমন নিচের সোর্স দুটি হতে পারে সঠিক দিক নির্দেশনাঃ

- Apple এর প্রকাশিত বই
- WWDC14 এর এ সম্পর্কিত কিছু ভিডিও

আর আমরা তো আছিই। আমাদের এই সিরিজের এবং [সম্ভাব্য বইয়ের](#) দ্বিতীয় সেকশনেই থাকছে বাংলায় ব্যাসিক সুইফ্ট লার্নিং এর উপর ১০টির বেশি চ্যাপ্টার। সিরিজের সব পোস্ট গুলোর এবং প্রিন্টেড বইয়ের আপডেট পেতে লাইক দিয়ে রাখুন আমাদের [ফেসবুকে ফ্যান পেজে](#)

আমাদের ব্লগ পোস্ট গুলোর চেয়ে অনেক বেশি বিস্তারিত আলোচনা, বিশ্লেষণ এবং কোড এক্সাম্পল থাকবে প্রিন্টেড বইয়ে।

পরের চ্যাপ্টারঃ পরের চ্যাপ্টারে থাকবে একটি সাধারণ প্রশ্ন যেটা অনেকেরই মনে জমে থাকে, "iOS এবং OSX এর অ্যাপ ডেভেলপমেন্টের জন্য Macbook, iMac, Mac mini অর্থাৎ Apple গ্যাজেট বাধ্যতামূলক কিনা" এর উপর আলোচনা এবং কিছু বিশ্লেষণ ও অবশ্যই কিছু বিকল্প ব্যবস্থার কথা।

Originally Posted [Here](#)