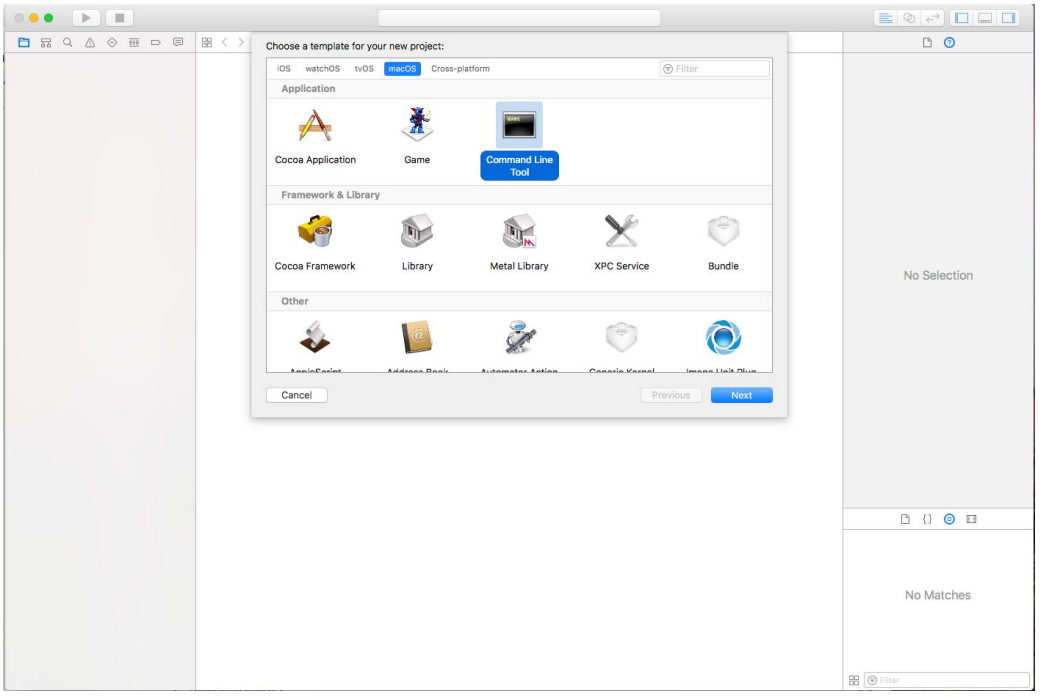
Урок 1

Знакомство с основами Objective-C и обзор среды Xcode

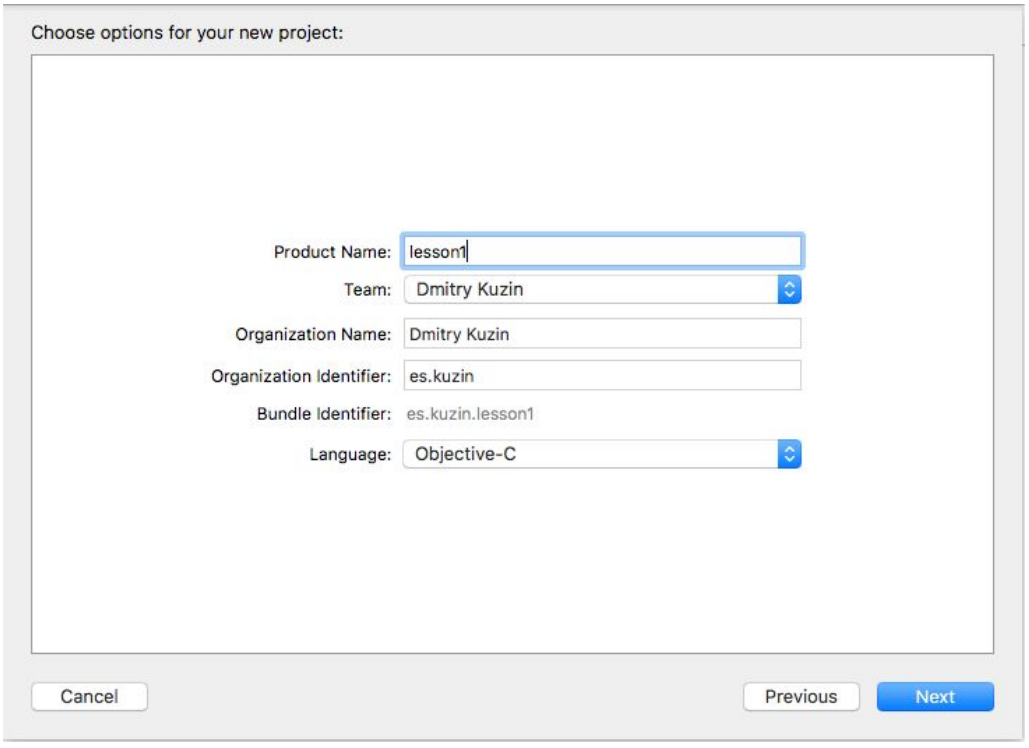
Знакомство с языком Objective-C. Изучение его предшественников. Отличия от других языков. Основные типы данных и арифметические операции. Обзор среды разработки Xcode. Организация файлов.

Обзор Xcode

Чтобы больше узнать об Xcode, рассмотрим создание консольного приложения. Для этого нажмем на **«Create a New Xcode Project»**, перейдем на вкладку **macOS** и выберем **«Command Line Tool»**.

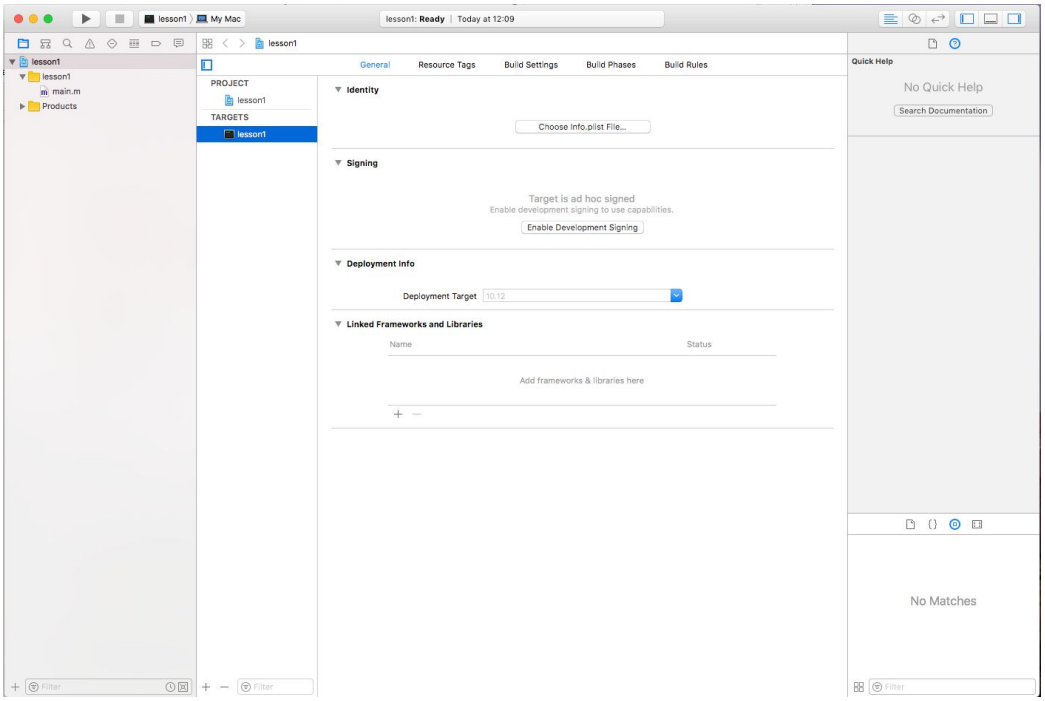


Укажем основные сведения о программе:

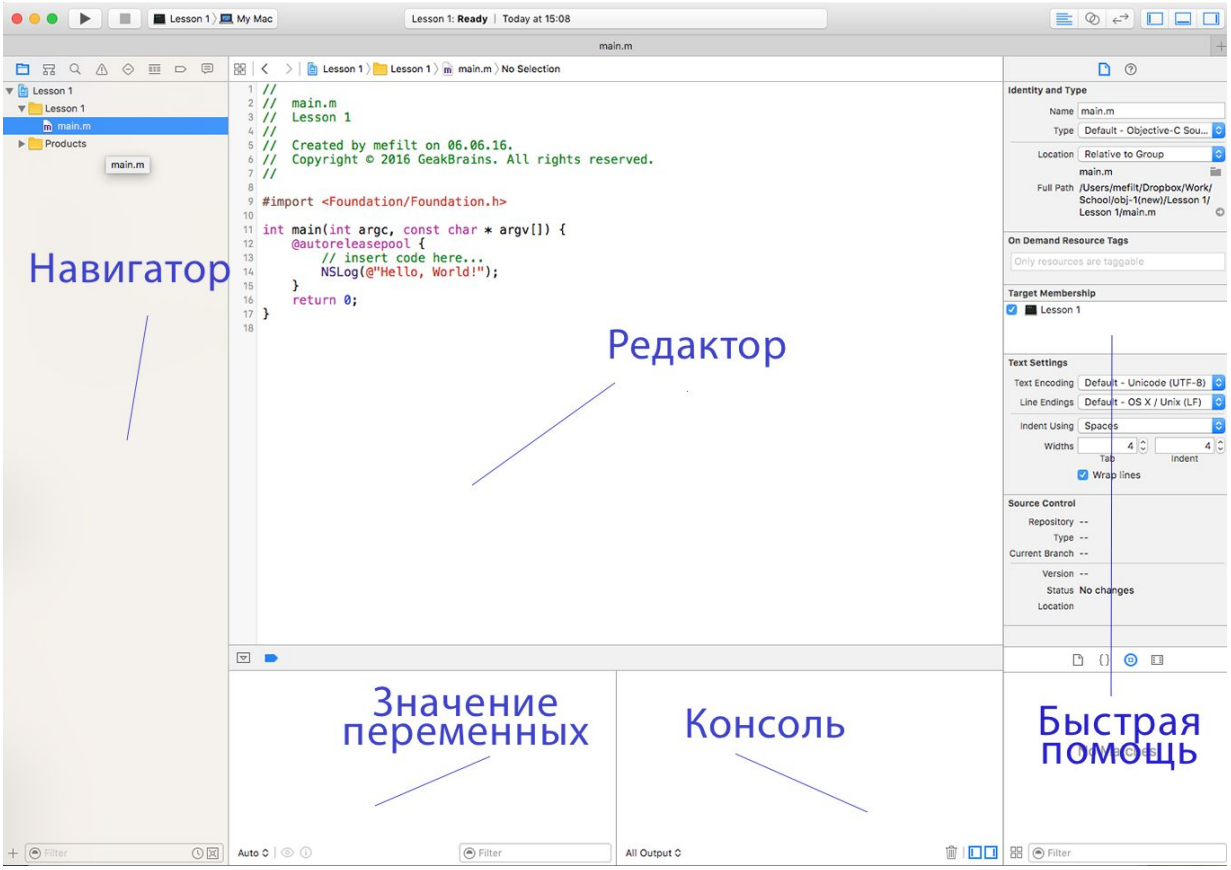


Здесь **Product Name** – название проекта; **Team** – имя аккаунта разработчика; **Organization Name** – название организации или лицо, которое создает программу; **Organization Identifier** – уникальный идентификатор, по которому можно будет однозначно определить программу.

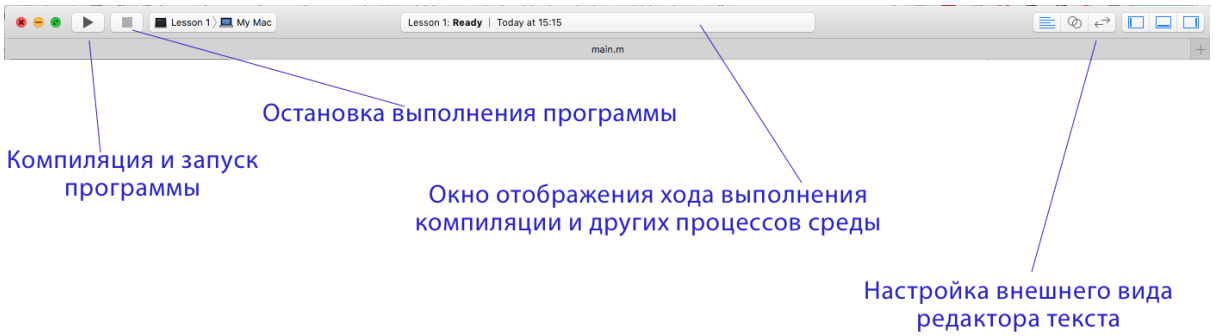
Далее выберем место для проекта и нажмем «Создать». Когда проект успешно создан, открывается основное окно проекта:



Основные составляющие среды:



И панель инструментов:



Файлы программы

Файлы Objective-C являются составными из заголовочного файла (.h) и файла реализации (.m). Заголовочный файл содержит интерфейс класса с описанием методов, свойств и переменных. Эти методы и переменные будут доступны в других классах при импорте данного заголовочного файла.

Файл реализации содержит в себе реализацию методов, описанных в заголовочных файлах. В него также могут входить иные методы, доступ к которым необходимо закрыть для иных классов.

Чтобы избежать путаницы, имена заголовочного файла и файла реализации идентичны.

Шаблонный код файла main.m

При создании программы по умолчанию добавляется файл **main.m**, который имеет следующий шаблонный код:

```
#import <Foundation/Foundation.h>

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

    }
    return 0;
}
```

Если раньше вы программировали на языке C, то вам знакомы объявление **main()** и инструкция **return (0)** в конце. Связано это с тем, что Objective-C – надстройка над C. Но есть еще и разница: например, **#import <Foundation/Foundation.h>**.

#import позволяет получить информацию о других классах на основе их заголовочных файлов. Эту директиву предоставляет компилятор, который программа Xcode использует для программ на языках Objective-C, C, C++. Также можно использовать **#include**. Это **#import** из языка C. Разница между ними состоит в том, что **#import** гарантирует, что заголовочный файл будет включен в программу только один раз, независимо количества обнаруженных в файле директив #import.

Знакомство с основами Objective-C

История Objective-C

Objective-C – это язык программирования, который является ядром операционных систем OS X и iOS. Он был создан значительно раньше этих ОС. Objective-C был разработан Бредом Коксом в 1980-х годах. Он объединил популярность и переносимость языка C с элегантностью SmallTalk. Первое крупное применение Objective-C состоялось при создании NeXTSTEP. Это объектно-ориентированная многозадачная операционная система на базе UNIX. Ее разработала компания NeXT, созданная Стивом Джобсом в 1985 году.

В 1996 году компания Apple приобрела NeXT, и ее продукт изменил название на привычное Cocoa. Это API для разработчиков было предоставлено компанией Apple абсолютно бесплатно.

С момента создания Objective-C и Cocoa прошло много лет, но они по-прежнему актуальны. Компания непрерывно развивала эти продукты, и они стали мощным набором инструментов разработки.

В Objective-C, как в надстройке над языком C, можно применять синтаксис C, функции и так далее.

Сравнение с языком Swift

Появление языка Swift не повлияло на популярность Objective-C. На нем продолжают писать приложения, а работодатели в крупных компаниях требуют от соискателей знания Objective-C. Дело в том, что Swift еще не так развит, и кодовая база предшественника куда более обширна.

Изучая Objective-C, можно освоить разработку приложений для платформ iOS и Mac OS «от истоков».

NSLog

Рассмотрим пример вывода строки в лог консоли:

```
NSLog(@"Hello World");
```

Эта строка кода выведет строку «Hello World» на консоль. Функция **NSLog()** является аналогом **printf()** в языке C. **NSLog()** принимает строку как первый аргумент, а также может применять другие аргументы для вывода дополнительных элементов. Например:

```
NSLog(@"Hello %@", @"World");
```

Результатом станет также «Hello World» в консоли. Таких дополнительных элементов может быть столько, сколько необходимо. Но для каждого их них в строке необходимо поставить специальный символ, соответствующий типу этого элемента (см. перечень ниже). С такими символами придется встречаться довольно часто, и по ходу курса они постепенно запомнятся.

Список спецификаторов формата NSString

Specifier	Description
%@	Objective-C object, printed as the string returned by `descriptionWithLocale:` if available, or `description` otherwise. Also works with `CFTypeRef` objects, returning the result of the `CFCopyDescription` function.
%%	'%' character.
%d, %D	Signed 32-bit integer (`int`).
%u, %U	Unsigned 32-bit integer (`unsigned int`).
%x	Unsigned 32-bit integer (`unsigned int`), printed in hexadecimal using the digits 0–9 and lowercase a–f.
%X	Unsigned 32-bit integer (`unsigned int`), printed in hexadecimal using the digits 0–9 and uppercase A–F.
%o, %O	Unsigned 32-bit integer (`unsigned int`), printed in octal.
%f	64-bit floating-point number (`double`).
%e	64-bit floating-point number (`double`), printed in scientific notation using a lowercase e to introduce the exponent.
%E	64-bit floating-point number (`double`), printed in scientific notation using an uppercase E to introduce the exponent.
%g	64-bit floating-point number (`double`), printed in the style of `%e` if the exponent is less than -4 or greater than or equal to the precision, in the style of `%f` otherwise.
%G	64-bit floating-point number (`double`), printed in the style of `%E` if the exponent is less than -4 or greater than or equal to the precision, in the style of `%f` otherwise.
%c	8-bit unsigned character (`unsigned char`).
%C	16-bit UTF-16 code unit (`unichar`).
%s	Null-terminated array of 8-bit unsigned characters. Because the `%s` specifier causes the characters to be interpreted in the system default encoding, the results can be variable, especially with right-to-left languages. For example, with RTL, `%s` inserts direction markers when the characters are not strongly directional. For this reason, it's best to avoid `%s` and specify encodings explicitly.
%S	Null-terminated array of 16-bit UTF-16 code units.
%p	Void pointer (`void *`), printed in hexadecimal with the digits 0–9 and lowercase a–f, with a leading `0x`.
%a	64-bit floating-point number (`double`), printed in scientific notation with a leading `0x` and one hexadecimal digit before the decimal point using a lowercase `p` to introduce the exponent.
%A	64-bit floating-point number (`double`), printed in scientific notation with a leading `0X` and one hexadecimal digit before the decimal point using a uppercase `P` to introduce the exponent.
%F	64-bit floating-point number (`double`), printed in decimal notation.

Вместо **NSLog** можно использовать вариант из языка C – **printf()**. Но **NSLog** предпочтительнее, так как позволяет выводить в лог консоли объекты Objective-C. Кроме этого, при вызове этой функции указывается дата и время, добавляется символ окончания строки – **\n**.

Префикс NS

В Objective-C можно часто встречается префикс **NS** (в том числе, в функции вывода в консоль NSLog). Он применяется для всех функций, констант и типов, которые взяты из среды Cocoa и однозначно определяет принадлежность к ней.

Главная задача префикса NS – предотвращать коллизии имен, возникающие при использовании одного и того же идентификатора для двух разных сущностей. Например, если бы функция NSLog не имела бы префикса и была бы просто Log, то программист мог создать другую функцию с таким же именем. Неизбежно возникли бы ошибки.

За буквами префикса NS скрывается предыдущее название разработчика – NeXTSTEP. После приобретения компании Apple решила не нарушать совместимость кода и продолжать использовать NS.

Так как префикс NS применяется средой Cocoa, то нежелательно использовать его для именования собственных классов. Это может ввести других программистов в заблуждение: они посчитают этот

класс компонентом Cocoa. В качестве префикса можно использовать собственные инициалы или название компании.

Синтаксические особенности

В этом курсе вам предстоит познакомиться с синтаксическими особенностями Objective-C, которые на первых порах могут быть непривычными. В примере вывода в консоль строки можно было заметить, что перед строкой стоит символ «@». Он помогает идентифицировать объекты (о них поговорим в следующих уроках). Запомним: перед строкой необходимо поставить этот символ.

Как и в C, а конце строки ставится двоеточие, чтобы компилятор однозначно смог определить завершение строки кода.

Еще одна синтаксическая особенность Objective-C – отправка сообщений (обращение к методу объекта). Для этого в квадратных скобках указывается объект и через пробел – его метод:

```
[object method];
```

Типы данных

Классификация базовых типов данных из языка C:

Тип	Размер	Диапазон значений
char	1 байт	От -127 до 127
bool	1 байт	true, false
short int	2 байта	От -32767 до 32767
unsigned short int	2 байта	От 0 до 65535
int	4 байта	От -32767 до 32767
unsigned int	4 байта	От 0 до 65535
long int	4 байта	От -2147483647 до 2147483647
unsigned long int	4 байта	От 0 до 4294967295
float	4 байта	От 1E-37 до 1E+37 с точностью не менее 6 значащих десятичных цифр
double	8 байт	От 1E-37 до 1E+37 с точностью не менее 10 значащих десятичных цифр
long double	10 байт	От 1E-37 до 1E+37 с точностью не менее 10 значащих десятичных цифр

Базовые типы Objective-C

Тип	Значение
BOOL	YES, NO
NSInteger	Целое число
CGFloat	Число с плавающей точкой
NSNumber	Объект, числового значения
NSString	Строка (@”Hello”)
NSMutableString (Изменяемая версия NSString)	Строка

В программировании тип данных – это система классификации, которая используется для указания возможных значений, принимаемых конкретным фрагментом данных. Тип данных указывает, как они представляются – как двоичные 1 и 0, когда они хранятся в памяти, а также как интерпретируются эти 1 и 0, когда они впоследствии считываются. Типы данных обеспечивают механизм указания операций и манипуляций, которые могут выполняться для значений определенного типа. Классифицируя данные в памяти, типы данных предоставляют способ присвоения значения более высокого уровня необработанным битам и байтам. Это дополнительное значение позволяет создавать приложения в более абстрактных и понятных терминах. Используя относительно небольшой набор типов данных, а также возможность создавать собственные типы, можно выражать и структурировать практически любую информацию.

Рассмотрим их характеристики.

BOOL хранит значение истинности и может выражаться только двумя состояниями: YES и NO.

NSInteger «отвечает» за целые числа (похож на примитивный тип int).

CGFloat хранит числа с плавающей точкой. Этот тип часто применяется для отображения размера и положения объектов на экране устройства (но это не единственное его применение).

NSNumber представляет числа (как целые, так и с плавающей запятой) в виде объекта. Одним из многочисленных применений для этого типа является представление чисел в массиве. Objective-C позволяет хранить в массивах только объекты, так что для добавления числа необходимо преобразовать его. Рассмотрим создание **NSNumber**:

```
NSInteger integer = 3;
// Создание NSNumber из NSInteger
NSNumber *integerNumber = [NSNumber numberWithInteger:integer];
// Создание NSNumber из BOOL
NSNumber *boolNumber = [NSNumber numberWithBool:NO];
// Создание NSNumber, используя литерал
NSNumber *number = @1;
// 3, 0, 1
NSLog(@"%@, %@, %@", integerNumber, boolNumber, number);
```

Таким образом в **NSNumber** преобразуются **NSInteger**, **BOOL**, примитивные типы. Также можно создавать объект, используя литерал.

NSString – это объект, который представляет строку. Рассмотрим пример создания строки:

```
NSNumber *number = @1;
NSString * string = [NSString stringWithFormat:@"%@", number];

NSString *anotherString = @"Hello";

NSLog(@"%@, %@", string, anotherString); // 1, Hello
```

Объект **NSString** можно создать, используя различные типы данных, применяя спецификаторы формата. Вариант с использованием литерала тоже применим.

Переменные

Чтобы создать переменную в Objective-C, надо последовательно указать ее тип, название, знак «=» и значение, которое необходимо ей присвоить. Иногда можно не указывать значение сразу: тогда изначально переменная будет равна одному из значений в зависимости от типа. Объект будет равен **nil**, целое число – **0**, число с плавающей точкой – **0.0**, тип **bool** будет иметь значение NO.

Пример создания переменных различных типов и вывода их значений в консоль:

```
int intValue = 10;
char *charValue = "s";
bool boolValue = false;
float floatValue = 1.2;
double doubleValue = 2.3;

BOOL boolObjc = YES;
NSInteger integer = 3;
CGFloat cgFloat = 3.1;
NSNumber *number = @1;
NSString * string = @"Hello";

NSLog(@"%d", intValue);       // 10
NSLog(@"%s", charValue);      // s
NSLog(@"%d", boolValue);      // 0
NSLog(@"%f", floatValue);     // 1.200000
NSLog(@"%f", doubleValue);    // 2.300000
NSLog(@"%d", boolObjc);       // 1
NSLog(@"%ld", (long)integer); // 3
NSLog(@"%f", cgFloat);        // 3.100000
NSLog(@"%@", number);         // 1
NSLog(@"%@", string);         // Hello
```

Пример создания переменных без значений и вывод в консоль:

```
int intValue;
char *charValue;
bool boolValue;
float floatValue;
double doubleValue;

BOOL boolObjc;
NSInteger integer;
CGFloat cgFloat;
NSNumber *number;
NSString * string;

NSLog(@"%d", intValue);     // 0
NSLog(@"%s", charValue);    // (null)
NSLog(@"%d", boolValue);    // 0
NSLog(@"%f", floatValue);   // 0.000000
NSLog(@"%f", doubleValue);  // 0.000000
NSLog(@"%d", boolObjc);     // 0
NSLog(@"%d", integer);      // 0
NSLog(@"%f", cgFloat);      // 0.000000
NSLog(@"%@", number);       // (null)
NSLog(@"%@", string);       // (null)
```

Константы

Константы – это переменные, которые не могут изменить значение после объявления. Их можно объявить двумя способами: воспользоваться ключевым словом **const** перед именем переменной или оператором препроцессора **#define**.

Пример объявления константы с помощью ключевого слова **const**:

```
NSString * const string = @"Hello";

NSLog(@"%@", string);
```

Соответственно, после объявления константы ***string*** изменить значение будет невозможно.

Чтобы освоить второй способ, рассмотрим понятие **#define**. Это оператор препроцессора, который в момент компиляции подставляет указанное значение в места, где она используется. Этот способ объявления глобальных переменных считается одним из самых простых в Objective-C.

Для создания константы с помощью **#define** необходимо указать ключевое слово, затем название, а после – само значение.

Пример реализации константы через оператор **#define**:

```
#define CONST 10

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

        NSLog(@"%d", CONST);

    }
    return 0;
}
```

В момент компиляции все вхождения слова CONST будут заменены на значение 10. Для разработчика этот процесс проходит полностью незамеченным.

Рассмотрим пример строковой константы, объявленной с помощью оператора **#define**:

```
#define STRING @"Hello"

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

        NSLog(@"%@", STRING);

    }
    return 0;
}
```

Необходимо задавать конкретные значения, которые заменяются посредством подстановок, выполняемых оператором **#define.**

Приведение типов

Чтобы получить переменную определенного типа из другого, можно воспользоваться приведением типов.

Допустим, что нам дано десятичное число, но необходимо, чтобы результат был в виде целого числа. Необходимо провести преобразование:

```
        double value = 1.2;

        int number = (int)value; // 1
```

В результате переменная ***number*** будет включать в себя целое число со значением 1.

В Objective-C существует и автоматическое приведение типов для любых числовых типов данных, кроме объектов. При присваивании значение иного типа будет приведено к типу переменной.

При сложении двух чисел разных типов их сумма будет соответствовать типу одного из них. Например, при сложении числа с плавающей точкой и целого результат будет приведет к числу с плавающей точкой. Математические операторы для приводимых типов и объектов применять нельзя.

Арифметические операции

Над переменными можно совершать арифметические операции: сложение, вычитание, умножение, деление и получение остатка от деления.

Пример сложения:

```
int number = 10 + 15; // 25
```

Пример вычитания:

```
int number = 15 - 10; // 5
```

Пример умножения:

```
int number = 10 * 15; // 150
```

Пример деления:

```
int number = 10 / 15; // 0
```

Пример остатка от деления:

```
int number = 10 % 2; // 0
```

Также в Objective-C присутствуют операторы инкремента и декремента, позволяющие увеличить или уменьшить число на 1. Для этого применяется следующая конструкция:

```
int a = 0;
int b = 1;

a++;
b--;
```

В результате выполнения переменная *a* будет иметь значение 1, а переменная *b* – 0.

Для увеличения или уменьшения числа можно применять конструкции вида: +=, -=, *=, /=.

Например:

```
 int a = 0;
int b = 1;

a += 2;
b -= 3;
```

В результате переменная *a* будет равна 2, а переменная *b* примет значение -2.

Практика

Задача 1. Создание программы, вычисляющей квадрат числа

Создаем новый проект. Выбираем и объявляем переменную, которая будет хранить значение, квадрат числа которого будет вычисляться. Затем производим расчет и выводим результат в лог:

```
        int number = 2;
        number = number * number;

        NSLog(@"%i", number); // 4
```

Улучшим этот пример. Вторую строчку кода, где производится расчет, можно удалить, а логику расчета перенести в функцию вывода в консоль:

```
  int number = 2;

        NSLog(@"%i", number * number); // 4
```

Задача 2. Создание программы для вывода списка одной строкой

Дано некоторое сопоставление имени и числа. Для более корректного отображения такой информации необходимо вывести ее одной строкой в консоль. Создадим 3 переменные с необходимыми значениями:

```
       int first = 10;
        int second = 20;
        int third = 30;
```

Вывод в консоль:

```
NSLog(@"First value - %i, Second value - %i, Third value - %i", first, second,
third);
```

Результатом выполнения программы станет строка, выведенная в консоль:

```
First value - 10, Second value - 20, Third value - 30
```

Задача 3. Создание программы, вычисляющей сумму введенных чисел

Для создания такой программы используем возможности языка C. Для получения введенного числа применим функцию ***scanf()***. Сначала создадим переменные, в которых будут храниться введенные значения:

```
        int first = 0;
        int second = 0;
```

После – выведем предложение о вводе цифр:

```
      printf("First number: ");
        scanf("%d", &first);
        printf("Second number: ");
        scanf("%d", &second);
```

По выполнении этих строк у программы уже будут необходимые значения, чтобы обработать результат:

```
NSLog(@"%d", first + second);
```

В результате получаем сумму введенных чисел:

```
First number: 1
Second number: 4
5
```

Практическое задание

1. Создать программу, которая будет применять к введенным числам различные арифметические операции (на основе практической задачи 3).
2. Улучшить программу: организовать вывод результата и переменных в консоль одной строкой (как в практической задаче 2).
3. *Создать приложение, которое будет вычислять среднее число из трех переменных, не применяя специальные функции.

Дополнительные материалы

1. Стивен Кочан. «Программирование на Objective-C».

2. Скотт Кнастер, Вакар Малик, Марк Далримпл.«Objective-C. Программирование для Mac OS X и iOS».

Используемая литература

Для подготовки данного методического пособия были использованы следующие ресурсы:

1. Стивен Кочан. «Программирование на Objective-C».
2. Скотт Кнастер, Вакар Малик, Марк Далримпл.«Objective-C. Программирование для Mac OS X и iOS».