Objective-C

Урок 2

Условные операторы и функции

Рассмотрение условных операторов в языке Objective-C. Реализация и применение функций.

Условные операторы

Условные операторы – важный элемент любого языка программирования. Они позволяют разработчику самостоятельно управлять потоком программы в зависимости от заданного условия. Условные операторы в языках C и Objective-C похожи.

Логические операторы

Благодаря логическим операторам можно проверить необходимое условие, сравнивая как переменные, так и обычные значения.

Логические операторы и примеры их использования

Оператор	Описание	Пример
<	Меньше, чем	A < B
>	Больше, чем	A > B
==	Равно	A == 10
!=	Не равно	A != 0
<=	Меньше или равно	A <= 3
>=	Больше или равно	A >= 4
&&	Логическое И	(A > B) && (A > 0)
||	Логическое ИЛИ	(A > B) || (A > 0)

Логические операторы можно использовать для формирования условия, но также их результат можно присваивать переменным. Рассмотрим пример:

```
BOOL flag = YES;
int a = 10;
int b = 20;

BOOL first = flag && a < b;          // YES
BOOL second = !flag && a <= b;       // NO
```

В результате первая переменная будет иметь значение YES, так как **flag** показывает истинность, и **a** действительно меньше **b**. Вторая переменная приняла значение **NO**, так как перед **flag** стоит знак отрицания («не»).

Структура оператора If

Оператор **If** – это один из самых популярных операторов условного ветвления. Его применение в Objective-C выглядит так же просто, как и в C:

```
if (/* Условие */) {
     // Оператор, который выполняется при истинности условия
}
```

При истинности условия, которое находится в скобках после ключевого слова *if,* будет выполнен код, который содержится внутри. Иначе программа проигнорирует этот участок.

Если необходимо проверить несколько условий, можно было бы использовать условные операторы так:

```
if (/* Условие */) {
    if (/* Условие 2 */) {
        // Оператор, который выполняется при истинности двух условий
    }
}
```

Но существует более правильный и элегантный подход, при котором оба условия указываются в условном операторе с применением ключевых слов. Здесь и будут удобны логические операторы.

Проверим два числа, чтобы первое было неотрицательным и больше второго:

```
int first = 10;
int second = 5;

if (first > second && first >= 0) {
      NSLog(@"Первое число больше второго и первое число неотрицательное");
}
```

Если изменить второе число так, чтобы оно было больше первого, то условие не будет выполнено. Но об ошибках необходимо сообщать пользователю. Для этого можно проверить противоположное условие, но правильнее – применить специальный оператор ***else***.

Реализация оператора if с else

Пример реализации условного оператора с оператором **else**:

```
if (/* Условие */) {
    // Условие выполнено
} else {
    // Условие не выполнено
}
```

Изменим пример должным образом, чтобы пользователь смог увидеть несоответствие условию:

```
        int first = 10;
        int second = 5;

        if (first > second && first >= 0) {
            NSLog(@"Первое число больше второго и первое число неотрицательное");
        } else {
            NSLog(@"Первое число меньше второго или первое число отрицательное");
        }
```

Теперь в результате выполнения программы при несоответствии условию будет выведено сообщение. Но точная причина возникновения ошибки останется неизвестной. Либо она возникла из-за того, что первое число меньше второго, либо потому, что первое число отрицательное. Чтобы уточнить, можно применить оператор ***elseif***. Условный оператор с применением **elseif** можно назвать условием с несколькими ветвями. Таких ветвей может быть много – в зависимости от количества условий, которые необходимо проверить.

Условный оператор с несколькими ветвями

Структура условного оператора с несколькими ветвями выглядит так:

```
if (/* Условие 1 */) {
    // Первое условие истинно
} else if (/* Условие 2 */) {
    // Второе условие истинно
} else {
    // Оба условия ложны
}
```

Преобразуем пример таким образом, чтобы можно было однозначно опознать ошибку:

```
if (first > second && first >= 0) {
    NSLog(@"Первое число больше второго и первое число неотрицательное");
} else if (first < 0) {
    NSLog(@"Первое число отрицательное");
} else {
    NSLog(@"Первое число меньше второго");
}
```

Оператор Switch

Оператор переключения, или **Switch**, позволяет выполнить блок кода при соответствии переменной определенному значению. Как правило, этот оператор применяется в качестве замены условному оператору с несколькими ветвями, если они многочисленны.

Структура оператора Switch

Структура оператора **Switch** выглядит так:

```
switch (/* Некоторое выражение или переменная */) {
    case /* Некоторое возможное значение */:
        /* Выражение или переменная равны значению */
        break;

    default:
        /* Выражение или переменная не равны ни одному значению */
        break;
}
```

Switch может принимать любое целочисленное значение (переменную) или выражение. Соответственно, возможные значения также должны быть целыми числами. Значения чисел не должны совпадать, а блоки кода при различных значениях могут повторяться.

Ключевое слово **break** необходимо для определения окончания блока кода, который выполняется при соответствии значению. Если его не указать, то программа будет выполнять все блоки кода, которые следуют после истинного значения.

Реализация оператора Switch

Рассмотрим пример. Необходимо проверить соответствие переменной определенному значению:

```
BOOL flag = YES;

switch (flag) {
    case YES:
        NSLog(@"Flag is TRUE");
        break;
    case NO:
        NSLog(@"Flag is FALSE");
        break;
}
```

Так как булевая переменная может иметь только два перечисленных значения, то нет необходимости указывать значение по умолчанию (***default***).

Оператор переключения перебирает все возможные варианты и обрабатывает сценарий, при котором значение равно выражению.

Тернарный оператор

Тернарный оператор – это еще один вид условного оператора, который позволяет вернуть необходимое значение в зависимости от условия. Таких значений может быть всего два: условие истинно или ложно.

Пример структуры тернарного оператора:

```
/* Условие */ ? /* Условие истинно */ : /* Условие ложно */;
```

Допустим, необходимо присвоить значение одной переменной в зависимости от истинности другой:

```
BOOL flag = YES;

int value = flag ? 500 : 300;
```

Переменной ***value*** будет присвоено значение в зависимости от значения ***flag***. В данном случае значение переменной будет равно 500.

Функции

Функция – это фрагмент программного кода, который выполняет определенное действие. К функциям можно обратиться из другого места программы. С их помощью можно улучшить код, избежать повторений. Функции в Objective-C можно реализовывать в таком же виде, как в языке C.

Функции могут применять параметры (атрибуты) и возвращать значение, а могут ничего не возвращать и не принимать. Чтобы функция возвращала значение, необходимо указать его тип. ***Void*** указывается для функций, которые ничего не возвращают.

Структура функции

Функция состоит из:

- Возвращаемого типа данных, если функция должна что-либо возвращать, иначе *void;*
- Имени функции;
- Параметров функции, если они необходимы;
- Тела функции, где выполняется необходимая логика.

Структура функции выглядит так:

```
/* Возвращаемый тип */ /* Имя функции */(/* Параметры функции */) {
    // Тело функции
    return /* Возвращаемое значение (если необходимо) */
}
```

Пример функции:

```
void function(int i) {
    NSLog(@"%i", i);
}
```

Данная функция ничего не возвращает, а принимает на вход значение типа **int**. При ее выполнении переданное значение будет выведено в консоль.

Функция с возвращаемым значением

Чтобы вызвать созданную функцию, необходимо просто указать ее имя и параметры в скобках. Так выглядит вызов параметра в файле **main.m**:

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

```
        function(10);

    }
    return 0;
}
```

В результате выполнения программы в консоль будет выведено число 10.

Рассмотрим пример реализации функции суммы и ее вызов:

```
int sum(int a, int b) {
    return a + b;
}

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

        int c = sum(10, 15);

        NSLog(@"%i", c);

    }
    return 0;
}
```

Была реализована функция **sum()** с параметрами **a** и **b**, которая возвращает целое число – сумму переданных чисел. В результате выполнения программы в консоль будет выведена сумма 10 и 15 (25).

Практика

Создание программы функции расчета и ее применение

Улучшим созданную нами функцию для расчета суммы двух чисел, чтобы она производила расчет в зависимости от передаваемого метода (сложение, вычитание, умножение, деление, остаток от деления).

Для этого понадобится передать один из методов в формате **NSString** (строки) и создать в функции проверку на метод.

Для проверки используем метод **NSString** – **isEqualToString**. В качестве параметра ему передается значение, с которым сравнивается строка.

Рассмотрим созданную функцию:

```
int calculate(NSString *method, int a, int b) {

    if ([method isEqualToString:@"+"]) {
        return a + b;
    }
```

```
    else if ([method isEqualToString:@"-"]) {
        return a - b;
    }
    else if ([method isEqualToString:@"*"]) {
        return a * b;
    }
    else if ([method isEqualToString:@"/"]) {
        return a / b;
    }
    else if ([method isEqualToString:@"%"]) {
        return a % b;
    }
    else {
        NSLog(@"Функция не знает переданный метод");
        return 0;
    }

    return a + b;

}
```

В результате выполнения функции будет проверен переданный параметр. Если он соответствует хотя бы одному поддерживаемому типу, то будет возвращен результат. Иначе будет возвращен 0, а в консоль выведено сообщение об ошибке.

После реализации функции необходимо применить ее в функции **main**:

```
 int main(int argc, const char * argv[]) {
    @autoreleasepool {
        int a = calculate(@"+", 2, 3);
        int b = calculate(@"-", 3, 2);
        int c = calculate(@"*", 4, 5);
        int d = calculate(@"/", 10, 2);
        int e = calculate(@"%", 6, 3);
        NSLog(@"Result: \n a = %i, \n b = %i, \n c = %i, \n d = %i, \n e = %i",
a, b, c, d, e);
    }
    return 0;
}
```

По выполнении программы в консоль будет выведено следующее:

```
Result:
 a = 5,
 b = 1,
 c = 20,
 d = 5,
 e = 0
```

Результат достигнут. При вызове функции и указании арифметического метода в формате строки и двух чисел, программа будет корректно рассчитывать необходимое значение.

Практическое задание

1. Создать функцию, которая будет проверять, входит ли переданная буква в английский алфавит.

2. Разделить метод **Calculate** (из практической задачи) на несколько методов (отдельно сложение, вычитание, умножение и деление).

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».