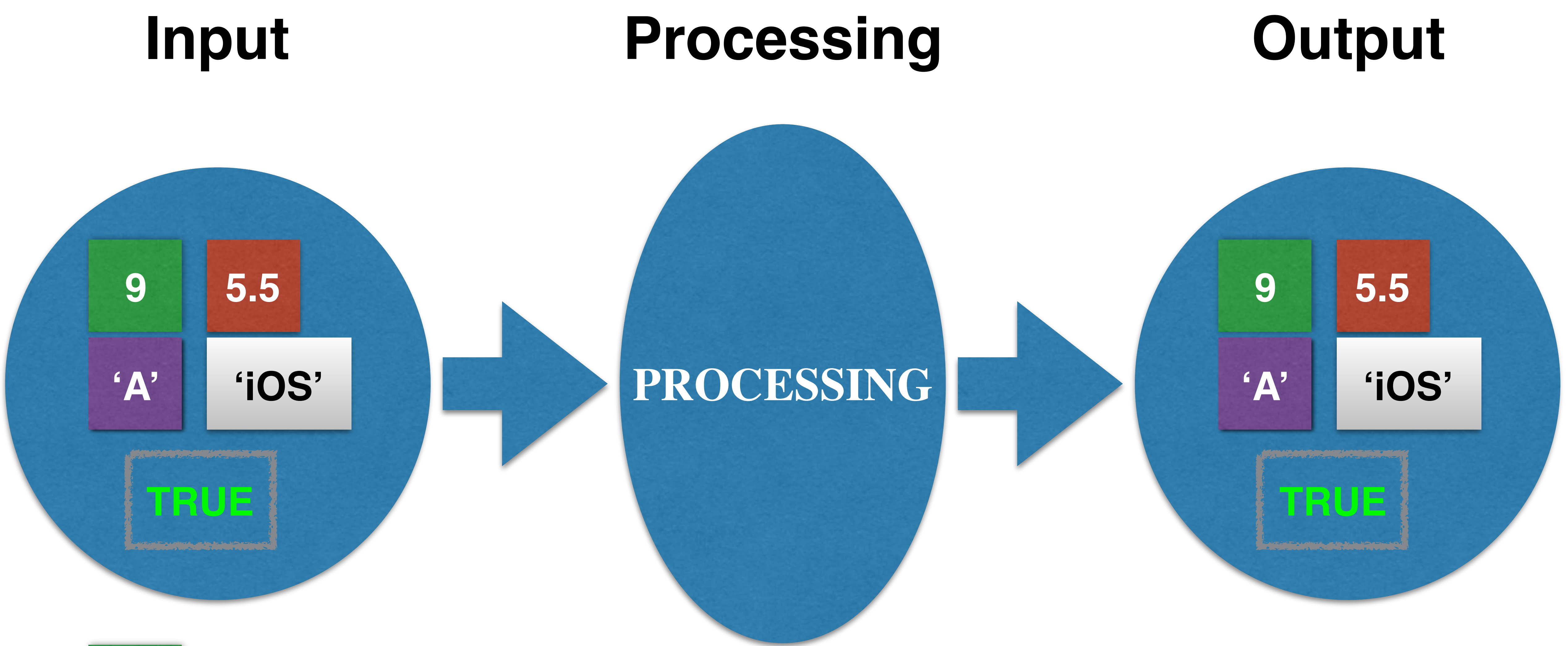
Day- 2,3

Datatypes, Operators, conditional statements, Control statements, structures, Enums, typedef, Preprocessor directives, I/O statements, Pointers

Objective-C (C + Smalltalk(OOPs))

Developed in 1980’s by BradCox and Tom Love.
iOS and MAC apps can be developed.

At WWDC 2014, Apple announced plans to replace Objective-C for Cocoa development with its new Swift language, which it characterized as "Objective-C without the C".



9	Whole values (int)
5.5	Floating values (float / double)
'A'	Single Character (char)
'iOS'	Multiple Characters (NSString)
true	TRUE/FALSE / 1/0 YES/NO (BOOL)

Basics

- Datatypes (int, long, float, double, char, BOOL)
- Input & Output Statements (scanf(),NSLog( ))
- Operators (Unary, Binary and Ternary)
- Conditional Statements (if, if-else and switch)
- Control Statements (for, while, do-while)
- Array (Group of similar datatype elements)
- Strings (Group of characters)
- Structures (struct)
- Enums (Enum)
- Preprocessor Directives (#define)
- Typecasting (Converting one datatype to another)

Datatypes

Every program needs some input data. The data can be of any type (whole numbers (int), floating point values (float), characters (char) etc…). That input data will be stored in a specified memory location allocated using datatypes.

Definition: Datatypes specifies to compiler that, what kind of values the declared variable can store.
or
Datatypes are used to allocate memory locations for the specific kind.
or
Datatype specifies to compiler that, what kind of memory location to be allocated for the declared variable. Then these declared variables only stores the same kind of value.

Syntax:

Datatype *variableName*; // Declaration
***variableName* = *someValue*; // Assignment**
Datatype *variableName* = *someValue*; // Initialisation

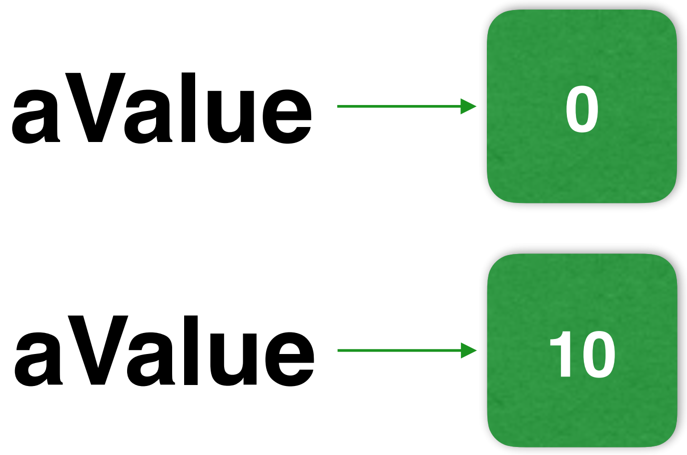
Datatypes available in Objective-C

int (1, 12, 123, 1234, 12345 etc)
long (1, 1212, 123456789 etc)
float (1.5, 2.9, 9.123456 etc)
double (1.12, 123.123, 1.123456789012345 etc)
char (‘A’, ‘B’… ‘Z’, ‘a’ … ‘z’ , ‘0’…’9’ !,@,#,$ etc)
BOOL (1, 0 , TRUE, FALSE, YES , NO)
id (any kind of Object but not primitives (which are above))
SEL (pointer to a method)

Variable Declaration Rules

- Variable name can not be a existing keywords such as (switch, break, continue, for, while etc…)
- Variable name must start with underscore or alphabet, no other special symbols are allowed
- No Spaces are allowed in variable name
- Two or more variables must not have the same name
- Once you declare a variable, you can modify the value of that variable any number of times.
- Use const for declaring a constant.

```
int int = 10;  // KEYWORD is not allowed
int 10Int = 10; // Started with NUMERIC
int my name = “Devendra”; // SPACES ARE NOT ALLOWED

int aVar = 10;
float aVar  = 20; // aVar is already declared
```



```
int aValue;
aValue = 10;
long lValue = 123456789;

float aFloat = 1.5;
double dValue = 12.121212211;

char cValue = ‘c';
BOOL aBool = TRUE;

Default values:
int, long, BOOL //0
float, double // 0.000000
char // ‘ ’

const int aConstant = 10;
aConstant = 20; // ERROR: constant can't be changed
```

Basic Program Syntax

```
int main(int argc, const char * argv[])
{
     // Variable Declarations(Storage Specifications)
     // Operation statements
    return 0;
}
```

Ex:

```
int main(int argc, const char * argv[])
{
    // Step 1
    int a = 10, b = 20;
    int c = a + b;

    return 0;
}
```

Output Statement

NSLog() is an output statement in Objective-C. The NSLog() function takes objective-c string as an input argument / parameter.

The input argument can be either plain string or formatted string.
Ex:

```
char *cString = "This is an C String”;
NSString *objcString = @"This is an Objc String”;

NSLog(@“This is a plain string”);
NSLog(@“I am attending iOS Course”);

NSLog(@“I am attending %i class”, 1); //
Here 1 is an integer value
int batchStartDate = 24;
NSLog(@“Batch started on  %i th”,
batchStartDate); // Batch started on 24th
```

Input Statement

scanf() is a function to read the input from the console.

Syntax:

scanf(“format Specifier”, addressOfVariable);

Ex: To read integer value

int a;

scanf(“%i”, &a);

// to read float

float aFloat;

scanf(“%f”, &aFloat);

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

    NSLog(@"Enter a value for inputVariable");
    scanf("%i", &inputVariable);

    NSLog(@"Entered value is : %i", inputVariable);

    return 0;
}
```

O/P:

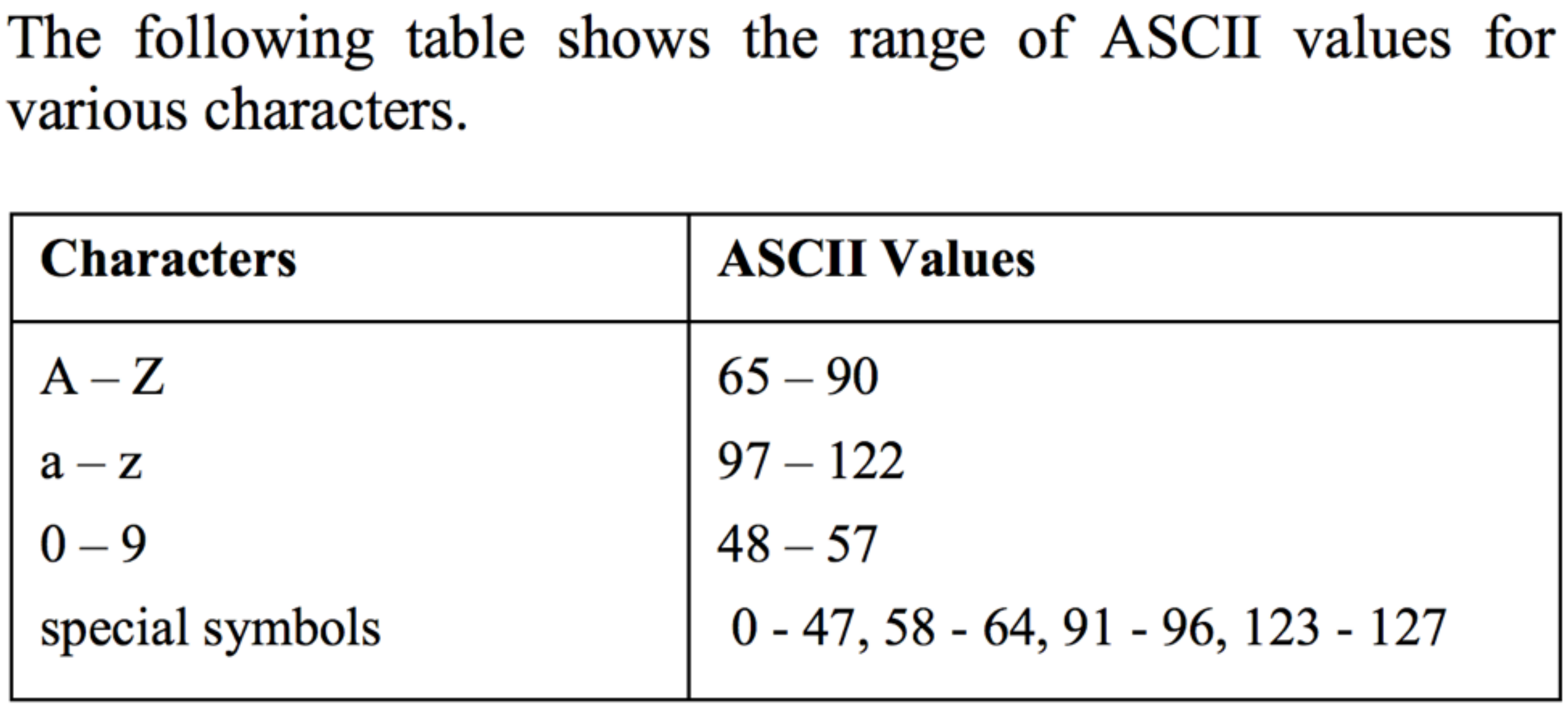
```
Enter a value for inputVariable
10
Entered value is: 10
```

Format Specifiers

Datatype	Format Specifier
int	%i or %d
long or NSInteger or NSUInteger	%li
char	%c
float	%f
double	%f
BOOL	%i
id	%@
Object	%@

ASCII

ASCII (American Standard Code for Information Interchange)
ASCII Values are encoded values for the symbols such as Alphabets, Numbers and Special Symbols (≈ ¥ ˆ ™ @ ‘ “ ^)



The following table shows the range of ASCII values for various characters.

Characters	**ASCII Values**
A – Z	65 – 90
a – z	97 – 122
0 – 9	48 – 57
special symbols	0 - 47, 58 - 64, 91 - 96, 123 - 127

Type Casting

Sometimes we are required to force the compiler to explicitly convert the value of an expression to a particular datatype. The process of converting one datatype to another datatype for the time being called Type Conversion.

Without type casting:

```
float result = 0;
int a = 6;
int b = 4;
result = a / b; // 6/4 = 1 ==>  1.000000
NSLog(@"Result of the above expression is: %f", result); // 1.000000
```

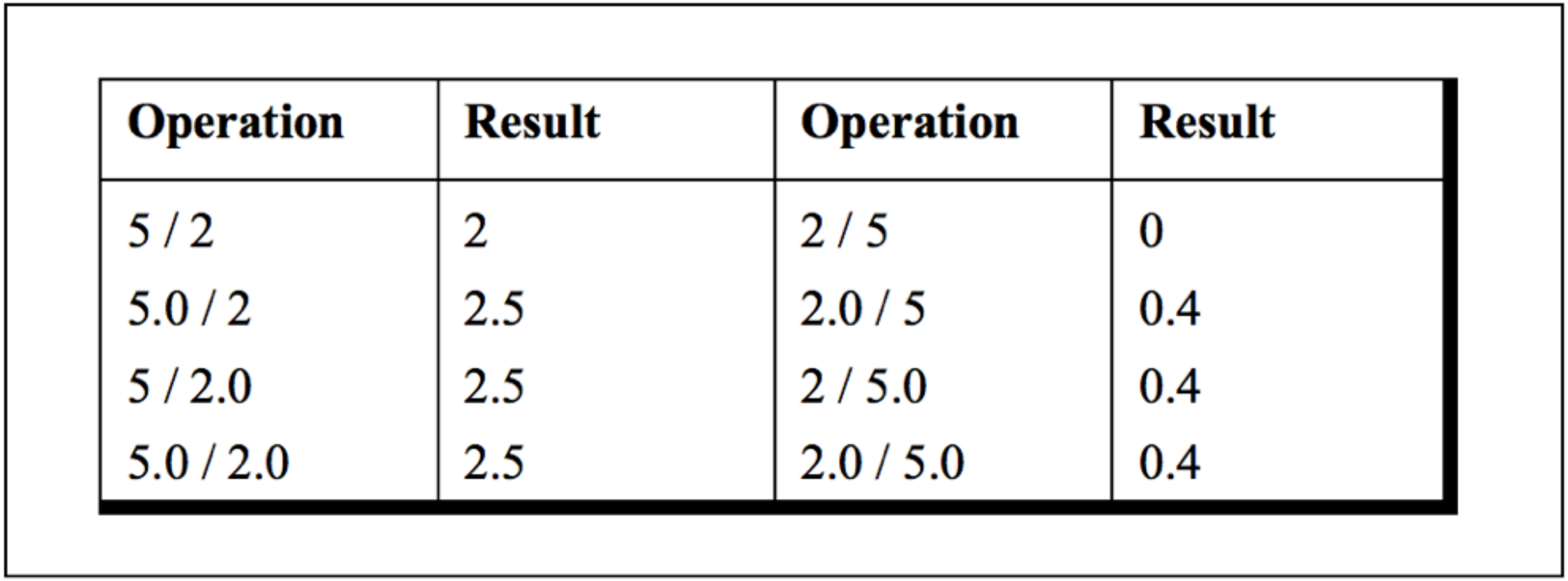
With type casting:

```
float result = 0;
int a = 6;
int b = 4;
result = (float)a / b; // 6.000000/4 = 1.5 ==>  1.500000
NSLog(@"Result of the above expression is: %f", result); // 1.500000
```

Implicit Type Conversion

In order to effectively develop C programs, it will be necessary to understand the rules that are used for the implicit conversion of floating point and integer values in C. These are mentioned below.

- An arithmetic operation between an integer and integer always yields an integer result.
- An operation between a real and real always yields a real result.
- An operation between an integer and real always yields a real result. In this operation the integer is first promoted to a real and then the operation is performed. Hence the result is real.



Operation	Result	Operation	Result
5 / 2	2	2 / 5	0
5.0 / 2	2.5	2.0 / 5	0.4
5 / 2.0	2.5	2 / 5.0	0.4
5.0 / 2.0	2.5	2.0 / 5.0	0.4

```
int a = 10;
float b = a; // 10.00000
b = 15.50;
a = b; // 15
a = ‘A’; // a = 65. ‘A’s ASCII value is 65
```

Operators

Definition: Operator is a symbol which performs operations on Operands. Operators are classified into 3 categories

i. Unary Operator: + , -, ++, —

An Operator, which performs operations on single operands

ii. Binary Operator:

An Operator, which performs operations on two operands

Arithmetic Operators: + , - , % , /, *

Compound Operators: +=, -=, %=,/= , *= , =

Comparison Operators: ==, >, <, >= , <=

Logical Operators: ||, &&, !=

iii. Ternary Operator: ?: (Conditional Operator)

An Operator, which performs operations on three operands

```
int aValue = 10;
int anotherInt = aValue;

NSLog(@"Unary - operation on aValue: %i", -aValue);
// -10

int incrementer = 10;

NSLog(@"Pre increment: %i", ++incrementer); // 11
NSLog(@"Post increment: %i", incrementer++); // 11
NSLog(@"Current value: %i", incrementer);    //12

NSLog(@"Pre Decrement: %i", --incrementer); //11
NSLog(@"Post Decrement: %i", incrementer--); //11
NSLog(@"Current value: %i", incrementer);  //10
```

```
// Arthematic Operators
NSLog(@"Arithmetic Sum: %i", aValue + anotherInt); // 20
NSLog(@"Arithmetic Subtract: %i", aValue - anotherInt); // 0
NSLog(@"Arithmetic Modulus: %i", aValue % anotherInt); // 0
NSLog(@"Arithmetic Division: %i", aValue / anotherInt); // 1
NSLog(@"Arithmetic multiply: %i", aValue * anotherInt); // 100

// Compound Operation
int a = 10, b = 20;
   a += b; // a = a + b;
   NSLog(@"%i",a); // 30
   a -= b; // a = a - b;
   NSLog(@"%i",a); // 10
   a %= b; // a = a % b;
   NSLog(@"%i",a); // 10
   a /= b; // a = a / b;
   NSLog(@"%i",a); // 0
   a *= b; // a = a * b;
   NSLog(@"%i",a); // 0
```

Conditional Statements (if, if-else)

```
Syntax:

if (<#condition#>)
{
    <#statements#> // Executes statements if
condition is TRUE
}


if (<#condition#>) {
    <#statements#> // Executes statements if
condition is TRUE

} else {
    <#statements#> // Executes statements if
condition is FALSE

}
```

```
// Conditional Statements
a = 10;
b = 20;

// If condition is always true for non zero values
if (a > b)
{
    NSLog(@"a is greater than b");
}

if (a > b)
{
    NSLog(@"A is greater than B");
    NSLog(@"Statement 2");
}
else
{
    NSLog(@"b is greaterthan a");
    NSLog(@"Statement 2");
}
```
NOTE: If condition becomes true only for non-zero expression result

Logical Operators(&&, ||, !)

Logical Operators are used to validate two or more expressions.

Logical And (&&): Becomes TRUE if all expressions are true.
Logical OR (||): Becomes TRUE if any one of the expressions are true
Logical Not (!): It negates the expression result

	&&	||	!
T,T	TRUE	TRUE	-
T,F	FALSE	TRUE	-
F,F	FALSE	FALSE	-
T	-	-	FALSE
F	-	-	TRUE

```
int main(int argc, const char * argv[])
{
    int a = 10, b = 20;

    if (a == 10 && b == 20)  //T && T = TRUE
    {
        NSLog(@"a is 10 and b is 20");
    }

    if (a == 10 && b == 10)  // T && F = FALSE
    {
        NSLog(@"a is 10 and b is 20");
    }

    if (a == 10 || b == 20)  //T || T =  TRUE
    {
        NSLog(@"a is 10 and b is 20");
    }

    if (a == 10 || b == 10)  //T || F = TRUE
    {
        NSLog(@"a is 10 and b is 20");
    }

    if (!(a == 0)) // !(F) = TRUE
    {
        NSLog(@"a is non zero");
    }
    return 0;
}
```

if-else ladder

```
if (<#condition1#>) {
    <#condition1 statements#>
} else if (<#condition2#>) {
    <#condition2 statements#>
} else {
    <#statements#>
}
```

```
int a = 10, b = 20, c = 30;
if (a > b && a > c){
    NSLog(@"A is largest");
} else if (b > c){
    NSLog(@"B is largest");
} else {
    NSLog(@"C is largest");
}

if (a > b && a > c){
    NSLog(@"A is largest");
} else {
    NSLog(@"C is largest");
} else if (b > c){
    NSLog(@"B is largest");
}
```
NOTE: else case must be the last case

```
int a = 10, b = 20, c = 30, d = 15;
if (a > b && a > c && a > d){
    NSLog(@"A is largest");
} else if (b > c && b > d){
    NSLog(@"B is largest");
} else if (c > d) {
    NSLog(@"C is largest");
}else{
    NSLog(@"D is largest");
}
```

Switch

Switch is a conditional statement, which is used to choose 1 choice / option among multiple options.

Ex:

Which school to choose

Which girl/boy to marry

To choose one among multiple options, we need to use multiple if / if-else statements. switch makes our life easier without using if, if else ladders.

Syntax:

```
switch (<#integer expression#>) {
case <#constant#>:
     <#statements#>
break;

case <#constant#>:
    <#statements#>
    break;
default:
    break;
}
```

Ex:

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

    int a = 1;

    switch (a) {
        case 1:
            NSLog(@"Expression value is: 1");
            break;
        case 2:
            NSLog(@"Expression value is: 2");
            break;
        case 3:
            NSLog(@"Expression value is: 3");
            break;
        default:
            NSLog(@"Expression result doesn't match with available 
cases");
            break;
    }
}
```

O/P:

```
Expression value is: 1
```

Tips

- The cases may or may not be in the order
- default case is not mandatory
- default case is executed if the provides cases doesn’t match.
- Case value must be an integer constant (floats are not allowed)
- break transfers the control to out of the block
- continue transfers the control to beginning of the loop
- Even if there are multiple statements to be executed in each case there is no need to enclose them within a pair of braces(unlike if, and else)

- We can execute same set of statements for multiple cases
- We can use Characters as case expressions. Actually those are converted and validated as ASCII values

```
char ch = 'b';
switch ( ch )
{
    case 'a' :
    case 'A' :
        printf ( "a as in ashar" ) ;
        printf ( "A as in Ashar" ) ;
        break ;
    case 'b' :
    case 'B' :
        printf ( "b as in brain" ) ;
        break ;
    case 'c' :
    case 'C' :
        printf ( "c as in cookie" ) ;
        break ;
    default :
        printf ( "wish you knew what are alphabets" ) ;
}
```

Control Statements

```
for (<#initialization#>; <#condition#>; <#increment#>)
{
    <#statements#>
}


while (<#condition#>) {
    <#statements#> // Execute statements if the 
condition is TRUE
}


do {
    <#statements#> //  Execute at least once 
irrespective of Condition
} while (<#condition#>);
```

```
for (int i = 0; i <= 10; i++)
{
    NSLog(@"i Value is : %i", i);
}

// TASK: Find a given value is even or odd
int t = 1;
if ( t % 2 ==0)
{
    NSLog(@"Given number is even");
}
else
{
    NSLog(@"Given number is Odd");
}
```

```
//Task-2 Check the given number is -Ve or +Ve

int t = 1;
if (t < 0)
{
    NSLog(@"-Ve");
}
else
{
    NSLog(@"+ve");
}

// TASK-3: check the given number is between 100 and 200
t = 150;
if ( t >= 100 && t <= 200)
{
    NSLog(@"Given value is between 100-200");
}
else
{
    NSLog(@"Given value is not between 100-200");
}
```

```
// Task-4: Check the given numbe is 100 or not
int t = 150;
if (t == 100)
{
    NSLog(@"T value is 100");
}

//Task-5: Printing even numbers between 1-10
for (int t = 1; t <= 10; t++)
{
    if ( t % 2 == 0)
    {
        NSLog(@"Given number: %i is even", t);
    }
    else
    {
        NSLog(@"Given number: %i is Odd", t);
    }
}

// Task-6: Print 8th table
int table = 8;
for (int i = 1; i <= 10; i++)
{
    NSLog(@"%i * %i = %i", table, i, table * i );
}
```

while & do-while

Task: Program to display identify all even and odd numbers b/w 1–10

```
int main(int argc, const char * argv[])
{
    int p = 1;
    while (p <=10)
    {

        if ( p % 2 == 0)
        {
            NSLog(@"%i is even", p);
        }
        else
        {
            NSLog(@"%i is Odd", p);
        }
        p++;
    }
    return 0;
}
```

O/P:

```
1 is odd
2 is even
3 is odd  . . . .
10 is even
```

```
int main(int argc, const char * argv[])
{
    int q = 1;
    do{
        if ( q % 2 == 0)
        {
            NSLog(@"%i is even", q);
        }
        else
        {
            NSLog(@"%i is Odd", q);
        }
        q++;

    }while (q <= 10);
}
```

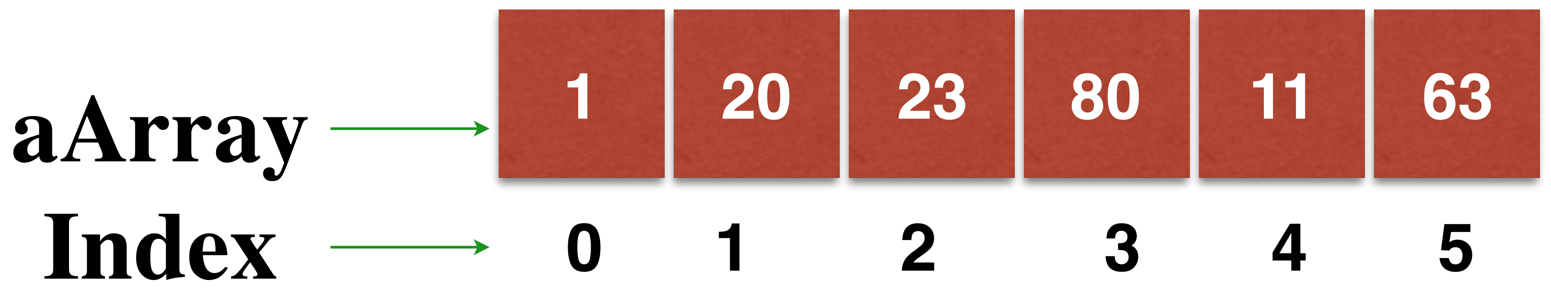
<u>O/P:</u>

```
1 is odd
2 is even
3 is odd  . . . .
10 is even
```

Arrays

Array is collection of Similar datatype values, those are stored in a continuous memory locations.

int aArray[10] = {1, 2, 3, 4, 5, 6}; // Dimension is not mandatory if initialisation takes place



- Each Memory location in the array are assigned a index.
- These indexes are starts from 0 to Array Size-1.
- These indexes are used to access the array elements.
- You can not have different datatype values in one array.
- Specifying the dimension is not mandatory if initialisation takes place
- Specifying dimension is mandatory if declaration takes place

Accessing Array Elements

Array elements can be accessed through the index of the array.

```
int a[5];
// assigning values
a[0] = 01;
a[1] = 10;
a[2] = 20;
a[3] = 30;
a[4] = 40;

// Accessing array elements
NSLog(@"1st element in a array is: %i", a[0]); // 01
NSLog(@"2nd element in a array is: %i", a[1]); // 10
NSLog(@"3rd element in a array is: %i", a[2]); // 20
NSLog(@"4th element in a array is: %i", a[3]); // 30
NSLog(@"5th element in a array is: %i", a[4]); // 40
```

Strings as Character Array

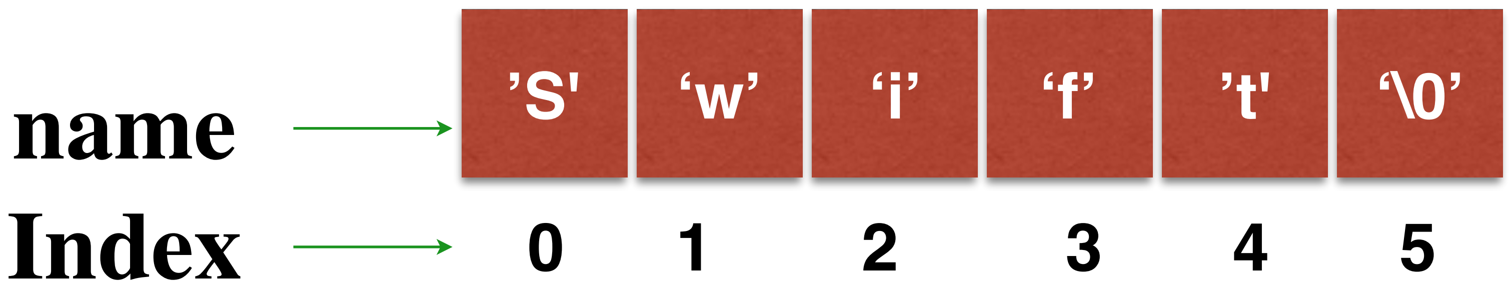
String is group of characters. Those can be represented by using character array.

Ex: “This is a C-String”

String constant is group of characters which is terminated with ‘\0’ character.

Ex:

```
char name[7] = “Swift";
```



Elements in the character array are accessed using index (0 to N−1)

Why the character arrays must be ended with ‘\0’?
Ans: Compiler uses this (null)character to identify the termination of the string.

Strings can also be represented using character pointer.

char *course = "iOS Programming";

Character Arrays vs Character Pointers

```
char d[] = "Hello";
d = "Hai"; //ERROR: Array type is not assignable

char *wish = "Hello";
wish = "Hai"; // Valid
```

Operations on character array

```
main()
{
    char name[ ] = “iOS Training";
    int i = 0 ;
    while(name[i] != '\0')
        {
            printf ( "%c", name[i] ) ;
            i++ ;
        }
}
```

Tasks

1. Write a program to read and calculate sum of N numbers?

```
int n = 0;
int sum = 0;
NSLog(@"Enter n value");
scanf("%i", &n);
for (int i = 0; i < n; i++)
{
    NSLog(@"Enter %i value", i+1);
    scanf("%i", &a[i]);
}
for (int i = 0; i < n; i++)
{
    sum += a[i];
}
NSLog(@"Sum of %i elements is : %i", n, sum);
```

- **Write a program to sort an array?**
- **Write a program to find the given element in a array or not?**
- **Write a program to reverse a string?**
- **Write a program to find out number of characters in a string?**
- **Write a program to print all ASCII values for the characters in a String?**
- **Write a program to find number of words in a String?**
- **Write a program to print all values in n-1 to 0 order?**

Structure

We have primitive datatypes to store integers, floats, characters, Booleans etc but we don't have any datatype to store complex data, such as Book information (Author, Pages, Price, Name etc). To create such Non Existing primitive datatypes, we use structure.

Structure is a collection of datatypes grouping together to create User defined datatype. struct is a keyword to create custom / user defined datatype.

Syntax:

```
struct NewDatatypeName
{
    // Properties declarations
};
```

```
struct BOOK {
    int pages;
    float price;
    char *name;
};

struct PEN {
    float price;
    char *brand;
}myPen;

int main(int argc, const char * argv[]) {
    @autoreleasepool
    {
        struct BOOK bookOne;
        bookOne.pages = 350;
        bookOne.price = 99.50;
        bookOne.name = “iOS”;

        myPen.brand = "Parker";
        myPen.price = 599.99;

        NSLog(@"Number of pages: %i",bookOne.pages );
        NSLog(@"Book Name: %s", bookOne.name);
        NSLog(@“Book Price: %f", bookOne.price);
    }
```

Enumerated Datatypes

- **Enumerated data type gives you an opportunity to invent your own data type and define values the variable of this data type can store**

- **This helps in making the program listing more readable**

- **Custom datatypes which are created by using enum are allows to store only the user specified values**

```
enum Fruit
{
    Apple, Pineapple, Mango, Banaga, Grape
};
enum WEEKEND
{
    SUNDAY, SATURDAY
};
enum WEEKDAY
{
    MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY
};
void main()
{
    int a = 10;
    enum  Fruit aFruit = Mango;
    enum WEEKDAY aWeekDay = FRIDAY;

    //  enum bWeekDay = SUNDAY; // SUNDAY is not part of WEEKDAY enum
declared
values

    if (aWeekDay == FRIDAY)
    {
        NSLog(@"Selected day is Friday");
    }
}
```

typedef

typedef is a keyword which is used to assign simpler names for an existing datatype.

```
typedef enum BOOK  ADVANCED_BOOK
typedef long  BIG_INT
```

ex:

```
BIG_INT  bigValue = 123456789;

ADVANCED_BOOK  bookOne;
bookOne.name = “Swift Programming”;
bookOne.price = 450.50;
bookOne.author = “Steve Jobs”;
```

Pre-processor Directives

The process of pre-compiling a program before the actual compilation takes place is known as Preprocessing. During preprocessing, preprocessor directives are compiled and replaced all parts of the program with actual values.

These preprocessor directives are preceded with #(hash or pound) symbol.

Ex:

```
#define  PI  3.145
#define  _numberOfStates 30
#define  _numberOfPlanets 7

#import “File2”
#import “File3”
```

Variable Scope

A *scope* in any programming is a region of the program where a defined variable can have it’s existence (access) and beyond that variable loses its’ availability / access.
There are three places where variables can be declared.

- **Global Scope**
- **Local Scope**
- **Block Scope**

Global Scope: A variable which is declared as global can be accessed in any part of the program / file.

Local Scope: A variable which is declared as local is only accessible with in that method.

Block Scope: A variable which is declared with in the block({…}) can be accessible only with in that block.

```
int aGlobalVariable = 10;
// NOTE: Global variable can be accessed in all parts of this file
int main()
{
    int aMainLocalVariable = 20;
    // NOTE: This variable is accessible only within this method
    NSLog(@"Accessing main method Local Variable: %i", aMainLocalVariable);
    NSLog(@"Accessing Global Variable: %i", aGlobalVariable);

    for (int blockVarA = 0; blockVarA <= 10; blockVarA++ )
    {
        int blockVarB = 15;
        NSLog(@"Accessing Block variable B: %i", blockVarB);
        NSLog(@"Accessing Block variable A: %i", blockVarA);

        NSLog(@"Accessing main method Local Variable: %i", aMainLocalVariable);
        NSLog(@"Accessing Global Variable: %i", aGlobalVariable);

        // NOTE: blockVarA and blockVarB are accessed only with in this for block
since they are declared in for block.
    }

    NSLog(@"Accessing Block variable A: %i", blockVarA); // ERROR, No Access
    NSLog(@"Accessing Block variable B: %i", blockVarB); // ERROR, No Access
}

void methodOne()
{
    NSLog(@"Accessing main method Local Variable: %i", aMainLocalVariable); //
ERROR: aMainLocalVariable is out of scope
    NSLog(@"Accessing Global Variable: %i", aGlobalVariable); // 10
}
```

Assume the output of the below statements

```
int a = 10;
int b = 15.99;
float f = 25.55;
char c = 70;

printf("%i", a);
printf("%i", b);
printf("%f", a);
printf("%c", b);
printf("%f", f);
printf(“%i", f);
printf("%c", c);
printf(“%i", c);
printf("%i", a/b);
printf("%i", a%b);
printf("%i", b%a);
printf("%f", b%a);
```

Please assume the output of the below statements

```
    int a = 10;
    int b = 15.99;
    float f = 25.55;
    char c = 70;

printf("%f", b%a);
printf("%f", (float)(b % a));
printf("%i", ++a);
printf("%i", a++);
printf("%i",a);
printf("%i",--a);
printf("%i",a--);
printf("%i",a);
printf("%i", -a);
printf("%i", a+=b);
printf("%f", a+=f);
printf("%i", a * b);
```

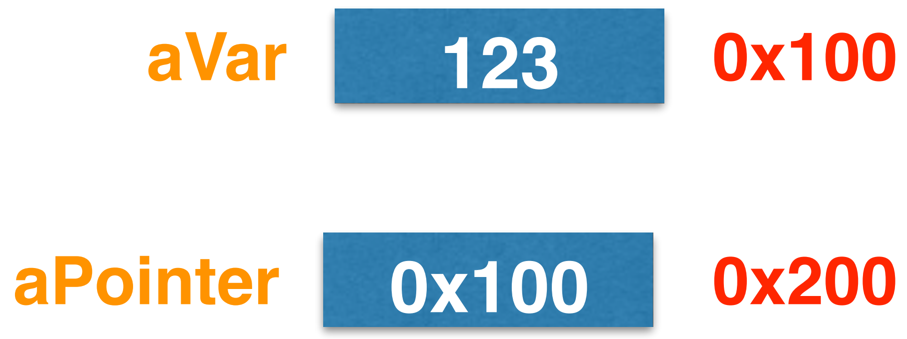
```
a = 0;
b = 15.99;
if (a)
{
    printf("%i",a);
}
if (a++)
{
    printf("%i",a);
}
if (a)
{
    printf("%i",a);
}
if (a || b)
{
    printf("%i , %i", a , b);
}
if (a && b)
{
    printf("TRUE");
}
for (int i = 0; i <= 10; i++)
{
    printf("%i", a);
}
for (int i = 0; i < 10; i++)
{
    printf("%i", i);
}
```

```
for (int i = 100; i < 100; i++)
{
    printf("%i", i);
}

for (int j = 10; j <= 100; j += 1)
{
    printf("%i", j);
}
```

Pointers

Pointer is a variable which stores the address of another variable of similar datatype.
Declare a pointer variable using *.

```
int  aVar = 123;

int *aPointer;
aPointer = &aVar;
```



```
NSLog(@"Value in aVar: %i", aVar); // 123

NSLog(@"Value in aPointer: %p", aPointer); //0x100

NSLog(@"Address of aPointer: %p", &aPointer); //0x200

NSLog(@"Address of aPointer: %p", &aVar); //0x100

NSLog(@"Value at address pointed by aPointer: %i",
*aPointer); //123
```

Functions / Methods

Definition:

- Function is a block of statements which performs some specific task.
- Some methods needs input arguments and some doesn't
- Some functions returns some value and some doesn't
- Specify void if method doesn't return value else specify proper return type
- Return type must be only one.
- Use return type when you need the result of the function further.

Syntax:

```
ReturnType methodName (Datatype InputArg1, Datatype inputArg2 , …)
{
    // Statements or Body of the method
    return returningVariable;
}
```

Ex:

```
void sayHi()
{
   NSLog(@“Hi”);
}

void wishWithMessage(char *wish)
{
    NSLog(@“%s World!”, wish);
}

void sum(int a, int b)
{
    NSLog(@“Sum is:  %i”,  a + b);
}

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

```objc
// Function Prototypes / Method Declarations
void sayHi();
void wishHim(char *wish);

int main(int argc, const char * argv[])
{
    @autoreleasepool
    {
        // Function Calling
        sayHi();
        wishHim("Hello");
    }
    return 0;
}

// Function Implementations
void sayHi()
{
    NSLog(@"Hi");
}

void wishHim(char *wish)
{
    NSLog(@"%s World!", wish);
}
```

Function Declaration:

A function which is not ended with semicolon

Ex: `void methodName();`

Function Definition / Implementation:

A function which has no semicolon and has open and close curly braces.

Ex:

```
void methodName()
{
   // Block of statements
}
```

Function Calling:

A statement which calls the implemented function known as Function calling

Ex:

```
   methodName();
```

Variety of Functions

- **Methods with no input (input arguments) and no output (Return type)**

```
void printHello()
{
   NSLog(@“Hello”);
}
```

- **Methods with input and no output**

```
void printValue(int a)
{
  NSLog(@“value is: %i”, a);
}
```

- **Methods with input and output**

```
int getASCIIvalue(char c);
{
     return c;
}
```

- **Methods with multiple input arguments**

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

```
void printSum(int aValue, int bValue)
{
    NSLog(@"Sum of Two values: %i", aValue + bValue);
}
```

// printSum(10, 20);

```
int multiply(int a, int b)
{
    return a * b;
}
```

// int mul = multiply(1,2);

```
void subtract(int one, int two)
{
    NSLog(@"Subtractions is: %i", one-two);
}
```

subtract(200, -10);

```
BOOL isEven(int someVar)
{
    if(someVar % 2 == 0)
    {
        return YES;
    }
    else
    {
        return NO;
    }
}
```

BOOL res = isEven(10);

```
int reversedNumber(int aNumber)
{
    int reversedNumber = 0;

    while(aNumber % 10 != 0)
    {
        reversedNumber = reversedNumber * 10 + aNumber % 10;
        aNumber = aNumber / 10;
    }
    return  reversedNumber;
}
```

int res = reversedNumber(123);

Call by value & Reference

By default values passed to a functions are values. Though you modify the input argument variables, those doesn’t reflect actual variables.

Passing variables’ address to a function as input arguments is known as Call By Reference. Modifying the values in those addresses reflects the actual variables’ values.

```
void callByValue(int inputArgOne, int inputArgTwo);
void callByReference(int *inputArgOne, int *inputArgTwo);

int main(int argc, const char * argv[])
{
    @autoreleasepool
    {
        int aVar = 10;
        int bVar = 20;
        callByValue(aVar, bVar);
        NSLog(@"aVar is: %i", aVar); // 10
        NSLog(@"bVar is: %i", bVar); // 20

        callByReference(&aVar, &bVar);
        NSLog(@"aVar is: %i", aVar); // 100
        NSLog(@"bVar is: %i", bVar); // 200
    }
    return 0;
}

void callByValue(int inputArgOne, int inputArgTwo)
{
    inputArgOne = 100;
    inputArgTwo = 200;
}
void callByReference(int *inputArgOne, int *inputArgTwo)
{
    *inputArgOne = 100;
    *inputArgTwo = 200;
}
```

Tasks

- Write a program which prints the following output
  - 1*n , 2*n, 3*n, 4*n, 5*n, 6*n, 7*n, … 10 times
  - 1, 1+2 , 1+2+3, 1+2+3+4, ….. till the given number
  - 1, -2, 3, -4, 5, -6 …. till the given number
  - -10, -9, -8 …. 10
  - 10 , 9 , 8 , …. 0
- Write a program which prints factorial of given number
- Write a program which reverses the given number
- Write a program which reverses the given string
- Write a program to find number of 1’s in a given number
- Write a program to find number of spaces in the string
- Write a program to find difference between sum of even and odd members
- Write a program to find the number if positive elements in an array
- Write a program to find sum of all numbers
- Write a program to check the given number existence
- Write a program to check the number of prime numbers in an array
- Write a program to sort an array

Tasks

- Write a method which displays all datatypes existing. (int, float, double, long, char, BOOL, id, SEL)
- write a method which prints the sum of 3 float values (33.50, 43.00, 23.50)
- Write a method which prints character for the given integer value
- (65 = A)
- Write a method which checks the given value is positive or negative
- (2: Even)
- Write a program which prints the table of given value (n*1=n …)
- Write a method which checks the given value is prime or not (3: Prime)
- Write a method which calculates the all digits sum of given value (123:6)
- Write a program which calculates all even digits sum of given value (12345: 6)
- Write a method to print all ASCII characters between 0 - 255
- Write a method which prints values between 50 -100
- Write a method which prints values between a - b
- Write a method calculate sum of values between a - b
- Write a method to verify the given value is Even or Odd

Tasks

- Write methods which prints the following output
  - 1*n , 2*n, 3*n, 4*n, 5*n, 6*n, 7*n, … 10 times
  - 1, 1+2 , 1+2+3, 1+2+3+4, ….. till the given number
  - 1, -2, 3, -4, 5, -6 …. till the given number
  - -10, -9, -8 …. 10
  - 10 , 9 , 8 , …. 0
- Write a method which prints factorial of given number
- Write a method which reverses the given number
- Write a method which reverses the given string
- Write a method to find number of 1’s in a given number
- Write a method to find number of spaces in the string
- Write a method to find difference between sum of even and odd members
- Write a method to find the number if positive elements in an array
- Write a method to find sum of all numbers
- Write a method to check the given number existence
- Write a method to check the number of prime numbers in an array
- Write a method to sort an array
- Create a structure with following elements and find sum of all book’s prices. BOOK(name, price, author)

Thank You

-DNReddy