



Welcome to Xcode

Version 8.2.1 (8C1002)

Get started with a playground

Explore new ideas quickly and easily.

Create a new Xcode project

Create an app for iPhone, iPad, Mac, Apple Watch or Apple TV.

Check out an existing project

Start working on something from an SCM repository.

[Open another project...](#)

■ The first line of the program `#include <stdio.h>` is a preprocessor

goScanDoc
~/Downloads/goscan/doc_principal_15Oct

DocScanImageProcessing
~/Documents/Workspace_C++

Learn_iOS
~/Documents/Learn_iOS/Unit_1

goScanDoc
...c/Nikunj_Work/goscan/doc_principal_15Oct

goScanDoc
...Doc/Nikunj_Work/GoScanApp_13Aug_2017

goScanDoc
...-goScanDoc/Rizky Work/goscan/doc_Rizky

goScanDoc
...ackup/Vikram_Work/tanmoy12-goscan/doc

HideCode_1
~/Documents/Workspace_C++

goScanDoc
...ikram_Work/CoreData_Marking/GoScanApp

[Open another project...](#)

Choose a template for your new project:

iOS watchOS tvOS macOS Cross-platform

Application



Cocoa



Game



Command Line
Tool

Framework & Library



Cocoa



Library



Metal Library



XPC Service



Bundle

Other



@



Robot



Cube



[Cancel](#)

[Previous](#)

[Next](#)

Choose options for your new project:

Product Name:

Team: Tanmoy Mondal

Organization Name: Tanmoy

Organization Identifier: Univ Tours

Bundle Identifier: Univ-Tours.ProductName

Language: C++

Choose options for your new project:

Product Name:

Team: Tanmoy Mondal

Organization Name: Tanmoy

Organization Identifier: Univ La Rochelle

Bundle Identifier: Univ-La-Rochelle.Appendre-Objective-C

Language: Objective-C

Première Programme en Objective-C

```
9 #import <Foundation/Foundation.h>
10
11 int main()
12 {
13     /* mon premier programme en Objective-C */
14     NSLog(@"Bonjour, Le Monde! \n");
15
16     return 0;
17 }
```



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

@interface SampleClass: NSObject
- (void)sampleMethod;
@end

@implementation SampleClass
- (void)sampleMethod{
    NSLog(@"Hello, World! \n");
}
@end

int main()
{
    /* my first program in Objective-C */
    SampleClass *sampleClass = [[SampleClass alloc] init];
    [sampleClass sampleMethod];
    return 0;
}
```

#import <Foundation/Foundation.h> : commande de préprocesseur, qui indique à un compilateur Objective-C d'inclure le fichier Foundation.h avant de procéder à la compilation proprement dite

@interface SampleClass: NSObject : montre comment créer une interface. Il hérite de NSObject, qui est la classe de base de tous les objets

(void)sampleMethod : montre comment déclarer une méthode

@end : marque la fin d'une interface

@implementation SampleClass : montre comment implémenter l'interface **SampleClass**

(void)sampleMethod{} : montre la mise en oeuvre de la méthode **sampleMethod**

@end : marque la fin d'une implémentation

main() : est la fonction principale où l'exécution du programme commence.

NSLog(...) : est une autre fonction disponible dans Objective-C qui provoque le message "Hello, World!" à afficher sur l'écran

return 0 : termine la fonction main () et renvoie la valeur 0

Définir constant

Avec **# define** préprocesseur

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

#define LENGTH 10
#define WIDTH 5
#define NEWLINE '\n'

int main()
{
    int area;

    area = LENGTH * WIDTH;
    NSLog(@"value of area : %d", area);
    NSLog(@"%@", NEWLINE);

    return 0;
}
```

Avec **const** motsclé

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

int main()
{
    const int LENGTH = 10;
    const int WIDTH = 5;
    const char NEWLINE = '\n';
    int area;

    area = LENGTH * WIDTH;
    NSLog(@"value of area : %d", area);
    NSLog(@"%@", NEWLINE);

    return 0;
}
```

Les structures de contrôle en C

Alternative:

if-else

Choix Multiple:

switch-case

Itérations:

for, while, do-while

Rupture de Contrôle:

break, continue, return

... **goto**

Les structures de contrôle en C

Les décisions - if then else

Pas de then en C

Le bloc " else "
est optionnel.

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

int main ()
{
    /* local variable definition */
    int a = 10;

    /* check the boolean condition using if statement */
    if( a < 20 )
    {
        /* if condition is true then print the following */
        NSLog(@"a is less than 20\n");
    }
    NSLog(@"value of a is : %d\n", a);

    return 0;
}
```

Les structures de contrôle en C

Les décisions – “if” then “else if”

Impliquée « if »

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

int main ()
{
    /* local variable definition */
    int a = 100;

    /* check the boolean condition */
    if( a < 20 )
    {
        /* if condition is true then print the following */
        NSLog(@"a is less than 20\n");
    }
    else
    {
        /* if condition is false then print the following */
        NSLog(@"a is not less than 20\n");
    }
    NSLog(@"value of a is : %d\n", a);

    return 0;
}
```

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

int main ()
{
    /* local variable definition */
    int a = 100;

    /* check the boolean condition */
    if( a == 10 )
    {
        /* if condition is true then print the following */
        NSLog(@"Value of a is 10\n");
    }
    else if( a == 20 )
    {
        /* if else if condition is true */
        NSLog(@"Value of a is 20\n");
    }
    else if( a == 30 )
    {
        /* if else if condition is true */
        NSLog(@"Value of a is 30\n");
    }
    else
    {
        /* if none of the conditions is true */
        NSLog(@"None of the values is matching\n");
    }
    NSLog(@"Exact value of a is: %d\n", a );

    return 0;
}
```

Les structures de contrôle en C

Imbriqué « if »

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

int main ()
{
    /* local variable definition */
    int a = 100;
    int b = 200;

    /* check the boolean condition */
    if( a == 100 )
    {
        /* if condition is true then check the following */
        if( b == 200 )
        {
            /* if condition is true then print the following */
            NSLog(@"Value of a is 100 and b is 200\n");
        }
    }
    NSLog(@"Exact value of a is : %d\n", a );
    NSLog(@"Exact value of b is : %d\n", b );

    return 0;
}
```

Les structures de contrôle en C

Les décisions – “switch”

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

int main ()
{
    /* local variable definition */
    char grade = 'B';

    switch(grade)
    {
        case 'A' :
            NSLog(@"%@", @"Excellent!\n");
            break;
        case 'B' :
        case 'C' :
            NSLog(@"%@", @"Well done\n");
            break;
        case 'D' :
            NSLog(@"%@", @"You passed\n");
            break;
        case 'F' :
            NSLog(@"%@", @"Better try again\n");
            break;
        default :
            NSLog(@"%@", @"Invalid grade\n");
    }
    NSLog(@"Your grade is %c\n", grade);

    return 0;
}
```

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

int main ()
{
    /* local variable definition */
    int a = 100;
    int b = 200;

    switch(a) {
        case 100:
            NSLog(@"%@", @"This is part of outer switch\n", a );
            switch(b) {
                case 200:
                    NSLog(@"%@", @"This is part of inner switch\n", a );
            }
            NSLog(@"%@", @"Exact value of a is : %d\n", a );
            NSLog(@"%@", @"Exact value of b is : %d\n", b );
    }

    return 0;
}
```

Les itérations

– While

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

int main ()
{
    /* local variable definition */
    int a = 10;

    /* while loop execution */
    while( a < 20 )
    {
        NSLog(@"%@", @"value of a: %d\n", a);
        a++;
    }

    return 0;
}
```

– for

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

int main ()
{
    /* for loop execution */
    int a;
    for( a = 10; a < 20; a = a + 1 )
    {
        NSLog(@"%@", @"value of a: %d\n", a);
    }

    return 0;
}
```

– do while

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

int main ()
{
    /* local variable definition */
    int a = 10;

    /* do loop execution */
    do
    {
        NSLog(@"%@", @"value of a: %d\n", a);
        a = a + 1;
    }while( a < 20 );

    return 0;
}
```

Imbriqué for

```
do
{
    statement(s);
    do
    {
        statement(s);
    }while( condition );
}while( condition );
```

```
for ( init; condition; increment )
{
    for ( init; condition; increment )
    {
        statement(s);
    }
    statement(s);
}
```

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

int main ()
{
    /* local variable definition */
    int i, j;

    for(i=2; i<100; i++) {
        for(j=2; j <= (i/j); j++)
            if(!(i%j)) break; // if factor found, not prime
        if(j > (i/j)) NSLog(@"%@", i);
    }

    return 0;
}
```

```
2 is prime
3 is prime
5 is prime
7 is prime
11 is prime
13 is prime
17 is prime
19 is prime
23 is prime
29 is prime
31 is prime
37 is prime
41 is prime
43 is prime
47 is prime
53 is prime
59 is prime
61 is prime
67 is prime
71 is prime
73 is prime
79 is prime
83 is prime
89 is prime
97 is prime
```

Infinité boucle for

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

int main ()
{
    for( ; ; )
    {
        NSLog(@"This loop will run forever.\n");
    }

    return 0;
}
```

Sortir de boucle

Break

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

int main ()
{
    /* local variable definition */
    int a = 10;

    /* while loop execution */
    while( a < 20 )
    {
        NSLog(@"%@", @"value of a: %d\n", a);
        a++;
        if( a > 15 )
        {
            /* terminate the loop using break statement */
            break;
        }
    }

    return 0;
}
```

Continue

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

int main ()
{
    /* local variable definition */
    int a = 10;

    /* do loop execution */
    do
    {
        if( a == 15 )
        {
            /* skip the iteration */
            a = a + 1;
            continue;
        }
        NSLog(@"%@", @"value of a: %d\n", a);
        a++;

    }while( a < 20 );

    return 0;
}
```

Fonctions

- **(retour_type) methode_nom:(argument_type1) argument_nom-1
joindre_argument2: (argument_type-2) argument_nom-2
joindre_argument3: (argument_type-3) argument_nom-3
joindre_argument4: (argument_type-4) argument_nom-4
.
.
.
.
.
joindre_argumentn: (argument_type-n) argument_nom-n**
{
.....
.....
.....
}

Déclarer un Fonction :

**(retour_type) methode_nom:(argument_type1) argument_nom-1
joindre_argument2: (argument_type-2) argument_nom-2.....
joindre_argumentn: (argument_type-n) argument_nom-n;**

-(int) max:(int)num1 andNum2:(int)num2;

Fonctions

Appeler par Valeur

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

@interface SampleClass:NSObject
/* method declaration */
- (int)max:(int)num1 andNum2:(int)num2;
@end

@implementation SampleClass

/* method returning the max between two numbers */
- (int)max:(int)num1 andNum2:(int)num2{
/* local variable declaration */
    int result;

    if (num1 > num2)
    {
        result = num1;
    }
    else
    {
        result = num2;
    }

    return result;
}

int main ()
{
    /* local variable definition */
    int a = 100;
    int b = 200;
    int ret;

    SampleClass *sampleClass = [[SampleClass alloc]init];

    NSLog(@"Before swap, value of a : %d\n", a );
    NSLog(@"Before swap, value of b : %d\n", b );

    /* calling a method to get max value */
    ret = [sampleClass max:a andNum2:b];
    NSLog(@"Max value is : %d\n", ret );

    return 0;
}
```

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

@interface SampleClass:NSObject
/* method declaration */
- (void)swap:(int)num1 andNum2:(int)num2;
@end

@implementation SampleClass

- (void)swap:(int)num1 andNum2:(int)num2
{
    int temp;

    temp = num1; /* save the value of num1 */
    num1 = num2; /* put num2 into num1 */
    num2 = temp; /* put temp into num2 */

}

int main ()
{
    /* local variable definition */
    int a = 100;
    int b = 200;

    SampleClass *sampleClass = [[SampleClass alloc]init];

    NSLog(@"Before swap, value of a : %d\n", a );
    NSLog(@"Before swap, value of b : %d\n", b );

    /* calling a function to swap the values */
    [sampleClass swap:a andNum2:b];

    NSLog(@"After swap, value of a : %d\n", a );
    NSLog(@"After swap, value of b : %d\n", b );

    return 0;
}
```

Appeler par Reference

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

@interface SampleClass:NSObject
/* method declaration */
- (void)swap:(int *)num1 andNum2:(int *)num2;
@end

@implementation SampleClass

- (void)swap:(int *)num1 andNum2:(int *)num2
{
    int temp;

    temp = *num1; /* save the value of num1 */
    *num1 = *num2; /* put num2 into num1 */
    *num2 = temp; /* put temp into num2 */

    return;
}

int main ()
{
    /* local variable definition */
    int a = 100;
    int b = 200;

    SampleClass *sampleClass = [[SampleClass alloc]init];

    NSLog(@"Before swap, value of a : %d\n", a );
    NSLog(@"Before swap, value of b : %d\n", b );

    /* calling a function to swap the values */
    [sampleClass swap:&a andNum2:&b];

    NSLog(@"After swap, value of a : %d\n", a );
    NSLog(@"After swap, value of b : %d\n", b );

    return 0;
}
```

Objective-C Numéros

+ (NSNumber *)numberWithBool:(BOOL)value

Crée et renvoie un objet NSNumber contenant une valeur donnée, en le traitant comme BOOL.

+ (NSNumber *)numberWithChar:(char)value

Crée et renvoie un objet NSNumber contenant une valeur donnée, en le traitant comme un caractère signé.

+ (NSNumber *)numberWithDouble:(double)value

Crée et renvoie un objet NSNumber contenant une valeur donnée, en le traitant comme un double.

+ (NSNumber *)numberWithFloat:(float)value

Crée et renvoie un objet NSNumber contenant une valeur donnée, en le traitant comme un float.

+ (NSNumber *)numberWithInt:(int)value

Crée et renvoie un objet NSNumber contenant une valeur donnée, en le traitant comme un int.

+ (NSNumber *)numberWithInteger:(NSInteger)value

Crée et renvoie un objet NSNumber contenant une valeur donnée, en le traitant comme un NSInteger.

- (BOOL)boolValue

Renvoie la valeur du destinataire sous forme de BOOL.

- (char)charValue

Renvoie la valeur du destinataire sous forme de char.

- (double)doubleValue

Renvoie la valeur du destinataire sous forme de double

- (float)floatValue

Renvoie la valeur du destinataire sous forme de float

- (NSInteger)integerValue

Renvoie la valeur du destinataire sous forme de NSInteger

- (int)intValue

Renvoie la valeur du destinataire sous forme de int

- (NSString *)stringValue

Renvoie la valeur du destinataire sous forme de string

Objective-C Numéros

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

@interface SampleClass: NSObject
- (NSNumber *)multiplyA:(NSNumber *)a withB:(NSNumber *)b;
@end

@implementation SampleClass
- (NSNumber *)multiplyA:(NSNumber *)a withB:(NSNumber *)b
{
    float number1 = [a floatValue];
    float number2 = [b floatValue];
    float product = number1 * number2;
    NSNumber *result = [NSNumber numberWithFloat:product];
    return result;
}

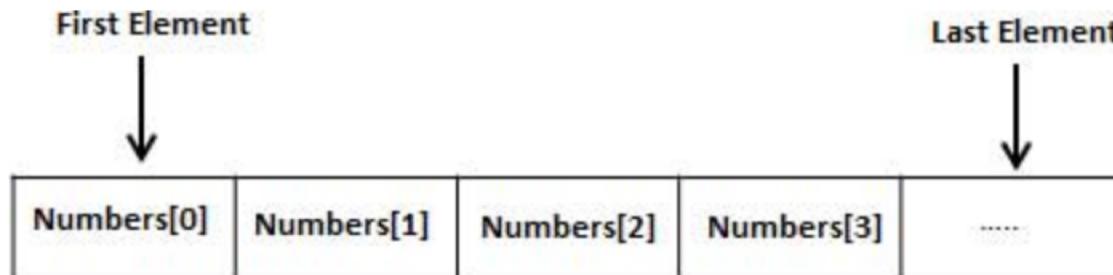
@end

int main()
{
    NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];

    SampleClass *sampleClass = [[SampleClass alloc] init];
    NSNumber *a = [NSNumber numberWithFloat:10.5];
    NSNumber *b = [NSNumber numberWithFloat:10.0];
    NSNumber *result = [sampleClass multiplyA:a withB:b];
    NSString *resultString = [result stringValue];
    NSLog(@"The product is %@",resultString);

    [pool drain];
    return 0;
}
```

Tableaux



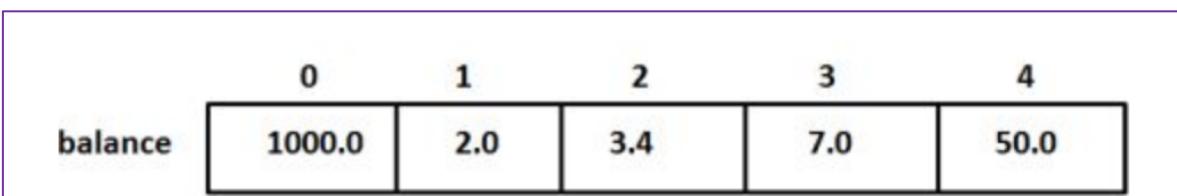
```
double balance[10];
```

OU

```
double balance[] = {1000.0, 2.0, 3.4, 7.0, 50.0};
```

OU

```
double balance[5] = {1000.0, 2.0, 3.4, 7.0, 50.0};
```



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

int main ()
{
    int n[ 10 ]; /* n is an array of 10 integers */
    int i,j;

    /* initialize elements of array n to 0 */
    for ( i = 0; i < 10; i++ )
    {
        n[ i ] = i + 100; /* set element at location i to i + 100 */
    }

    /* output each array element's value */
    for ( j = 0; j < 10; j++ )
    {
        NSLog(@"Element[%d] = %d\n", j, n[j] );
    }

    return 0;
}
```

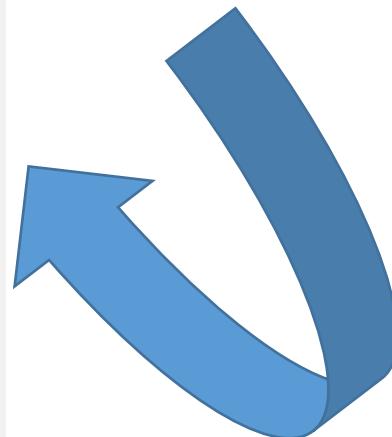
Tableaux Multidimensionnelle

	Column 0	Column 1	Column 2	Column 3
Row 0	a[0][0]	a[0][1]	a[0][2]	a[0][3]
Row 1	a[1][0]	a[1][1]	a[1][2]	a[1][3]
Row 2	a[2][0]	a[2][1]	a[2][2]	a[2][3]

```
int a[3][4] = {  
    {0, 1, 2, 3} , /* initialiseurs pour l'index de ligne par 0 */  
    {4, 5, 6, 7} , /* initialiseurs pour l'index de ligne par 1 */  
    {8, 9, 10, 11} /* initialiseurs pour l'index de ligne par 2 */  
}
```

```
a[0][0]: 0  
a[0][1]: 0  
a[1][0]: 1  
a[1][1]: 2  
a[2][0]: 2  
a[2][1]: 4  
a[3][0]: 3  
a[3][1]: 6  
a[4][0]: 4  
a[4][1]: 8
```

```
#import <Foundation/Foundation.h>  
  
int main ()  
{  
    /* an array with 5 rows and 2 columns*/  
    int a[5][2] = { {0,0}, {1,2}, {2,4}, {3,6},{4,8} };  
    int i, j;  
  
    /* output each array element's value */  
    for ( i = 0; i < 5; i++ )  
    {  
        for ( j = 0; j < 2; j++ )  
        {  
            NSLog(@"%@", a[i][j]);  
        }  
    }  
    return 0;  
}
```



Passer un tableaux dans un fonctions

Technique -1

```
- (void) myFunction(int *) param  
{  
    :  
    :  
}
```

```
-(double) getAverage:(int []) arr andSize:(int) size  
{  
    int i;  
    double avg;  
    double sum;  
  
    for (i = 0; i < size; ++i)  
    {  
        sum += arr[i];  
    }  
  
    avg = sum / size;  
  
    return avg;  
}
```

Technique -2

```
- (void) myFunction(int [10] )param  
{  
    :  
    :  
}
```

```
#import <Foundation/Foundation.h>  
  
@interface SampleClass: NSObject  
  
/* function declaration */  
-(double) getAverage:(int []) arr andSize:(int) size;  
  
@end  
  
@implementation SampleClass  
  
-(double) getAverage:(int []) arr andSize:(int) size  
{  
    int i;  
    double avg;  
    double sum =0;  
  
    for (i = 0; i < size; ++i)  
    {  
        sum += arr[i];  
    }  
  
    avg = sum / size;  
  
    return avg;  
}  
  
@end  
  
int main ()  
{  
    /* an int array with 5 elements */  
    int balance[5] = {1000, 2, 3, 17, 50};  
    double avg;  
  
    SampleClass *sampleClass = [[SampleClass alloc]init];  
    /* pass pointer to the array as an argument */  
    avg = [sampleClass getAverage:balance andSize: 5] ;  
  
    /* output the returned value */  
    NSLog( @"Average value is: %f ", avg );  
  
    return 0;  
}
```

Technique -3

```
-(void) myFunction: (int []) param  
{  
    :  
    :  
}
```

Retourner un tableau ou pointer de fonction

- ne peut pas renvoyer un tableau entier en tant qu'argument à une fonction
- peut renvoyer un pointeur vers un tableau en spécifiant le nom du tableau sans index
- Ne préférer pas renvoyer le adresse de un variable locale dehors de un fonction
- On doit déclarer une variable locale comme un **static** variable

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

@interface SampleClass: NSObject
- (int *) getRandom;
@end

@implementation SampleClass

/* function to generate and return random numbers */
- (int *) getRandom
{
    static int r[10];
    int i;

    /* set the seed */
    srand( (unsigned)time( NULL ) );
    for ( i = 0; i < 10; ++i)
    {
        r[i] = rand();
        NSLog( @"r[%d] = %d\n", i, r[i]);
    }

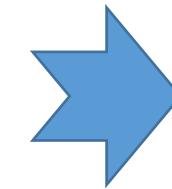
    return r;
}

@end

/* main function to call above defined function */
int main ()
{
    /* a pointer to an int */
    int *p;
    int i;

    SampleClass *sampleClass = [[SampleClass alloc] init];
    p = [sampleClass getRandom];
    for ( i = 0; i < 10; i++ )
    {
        NSLog( @"*(p + %d) : %d\n", i, *(p + i));
    }

    return 0;
}
```



```
r[0] = 1484144440
r[1] = 1477977650
r[2] = 582339137
r[3] = 1949162477
r[4] = 182130657
r[5] = 1969764839
r[6] = 105257148
r[7] = 2047958726
r[8] = 1728142015
r[9] = 1802605257
*(p + 0) : 1484144440
*(p + 1) : 1477977650
*(p + 2) : 582339137
*(p + 3) : 1949162477
*(p + 4) : 182130657
*(p + 5) : 1969764839
*(p + 6) : 105257148
*(p + 7) : 2047958726
*(p + 8) : 1728142015
*(p + 9) : 1802605257
```

Pointeur vers un tableau

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

int main ()
{
    /* an array with 5 elements */
    double balance[5] = {1000.0, 2.0, 3.4, 17.0, 50.0};
    double *p;
    int i;

    p = balance;

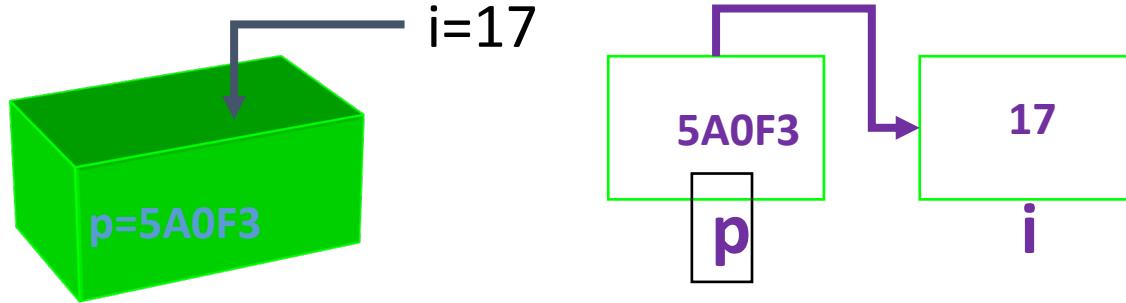
    /* output each array element's value */
    NSLog( @"Array values using pointer\n");
    for ( i = 0; i < 5; i++ )
    {
        NSLog(@"%@", *(p + i));
    }

    NSLog(@"Array values using balance as address\n");
    for ( i = 0; i < 5; i++ )
    {
        NSLog(@"%@", *(balance + i));
    }

    return 0;
}
```

```
Array values using pointer
*(p + 0) : 1000.000000
*(p + 1) : 2.000000
*(p + 2) : 3.400000
*(p + 3) : 17.000000
*(p + 4) : 50.000000
Array values using balance as address
*(balance + 0) : 1000.000000
*(balance + 1) : 2.000000
*(balance + 2) : 3.400000
*(balance + 3) : 17.000000
*(balance + 4) : 50.000000
```

Les pointeurs, c'est quoi?



```
1 int *ip; /* pointeur vers un integer */
2 double *dp; /* pointeur vers un double */
3 float *fp; /* pointeur vers un float */
4 char *ch /* pointeur vers un character */
```

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

int main ()
{
    int var = 20; /* actual variable declaration */
    int *ip; /* pointer variable declaration */

    ip = &var; /* store address of var in pointer variable*/
    NSLog(@"Address of var variable: %x\n", &var );

    /* address stored in pointer variable */
    NSLog(@"Address stored in ip variable: %x\n", ip );

    /* access the value using the pointer */
    NSLog(@"Value of *ip variable: %d\n", *ip );

    return 0;
}
```

Déclaration de Pointeurs

Le symbole ***** est utilisé entre le type et le nom du pointeur

- Déclaration d'un entier: **int i;**
- Déclaration d'un pointeur vers un entier: **int *p;**

Exemples de déclarations de pointeurs

```
int *pi; /* pi est un pointeur vers un int
           *pi désigne le contenu de l'adresse */

float *pf; /* pf est un pointeur vers un float */

char c, d, *pc; /* c et d sont des char*/
                  /* pc est un pointeur vers un char */

double *pd, e, f; /* pd est un pointeur vers un double*/
                   /* e et f sont des doubles */

double **tab; /* tab est un pointeur pointant sur un pointeur qui
                 pointe sur un flottant double */
```

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

int main ()
{
    int *ptr = NULL;

    NSLog(@"The value of ptr is : %x\n", ptr );

    return 0;
}
```

Tableaux de pointers

Est que on a bien compris le tableaux ?

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

const int MAX = 3;

int main ()
{
    int var[] = {10, 100, 200};
    int i;

    for (i = 0; i < MAX; i++)
    {
        NSLog(@"%@", var[i] );
    }
    return 0;
}
```

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

const int MAX = 4;

int main ()
{
    char *names[] = {
        "Zara Ali",
        "Hina Ali",
        "Nuha Ali",
        "Sara Ali",
    };

    int i = 0;

    for ( i = 0; i < MAX; i++)
    {
        NSLog(@"%@", names[i] );
    }
    return 0;
}
```

Alors pointer de tableaux ensuite !!

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

const int MAX = 3;

int main ()
{
    int var[] = {10, 100, 200};
    int i, *ptr[MAX];

    for ( i = 0; i < MAX; i++)
    {
        ptr[i] = &var[i]; /* assign the address of integer. */
    }
    for ( i = 0; i < MAX; i++)
    {
        NSLog(@"%@", *ptr[i] );
    }
    return 0;
}
```

Arithmétique du pointer

Incrémenter un pointeur

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

const int MAX = 3;

int main ()
{
    int var[] = {10, 100, 200};
    int i, *ptr;

    /* let us have array address in pointer */
    ptr = var;
    for ( i = 0; i < MAX; i++)
    {

        NSLog(@"%@", @"Address of var[%d] = %x\n", i, ptr );
        NSLog(@"%@", @"Value of var[%d] = %d\n", i, *ptr );

        /* move to the next location */
        ptr++;
    }
    return 0;
}
```

Décrémenter un pointeur

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

const int MAX = 3;

int main ()
{
    int var[] = {10, 100, 200};
    int i, *ptr;

    /* let us have array address in pointer */
    ptr = &var[MAX-1];
    for ( i = MAX; i > 0; i--)
    {

        NSLog(@"%@", @"Address of var[%d] = %x\n", i, ptr );
        NSLog(@"%@", @"Value of var[%d] = %d\n", i, *ptr );

        /* move to the previous location */
        ptr--;
    }
    return 0;
}
```

Comparaisons du pointeur

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

const int MAX = 3;

int main ()
{
    int var[] = {10, 100, 200};
    int i, *ptr;

    /* let us have address of the first element in pointer */
    ptr = var;
    i = 0;
    while ( ptr <= &var[MAX - 1] )
    {

        NSLog(@"%@", @"Address of var[%d] = %x\n", i, ptr );
        NSLog(@"%@", @"Value of var[%d] = %d\n", i, *ptr );

        /* point to the previous location */
        ptr++;
        i++;
    }
    return 0;
}
```

Envoyer un pointer dans un fonction en C

Passer pointer dans un fonction

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

@interface SampleClass:NSObject
- (void) getSeconds:(int *)par;

@end
@implementation SampleClass

- (void) getSeconds:(int *)par{
/* get the current number of seconds */
    *par = time( NULL );
    return;
}

@end

int main ()
{
    int sec;

    SampleClass *sampleClass = [[SampleClass alloc] init];
    [sampleClass getSeconds:&sec];

    /* print the actual value */
    NSLog(@"Number of seconds: %d\n", sec );

    return 0;
}
```

Passer pointer de tableaux dans un fonction

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

@interface SampleClass:NSObject
/* function declaration */
- (double) getAverage:(int *)arr ofSize:(int) size;
@end

@implementation SampleClass

- (double) getAverage:(int *)arr ofSize:(int) size
{
    int i, sum = 0;
    double avg;

    for (i = 0; i < size; ++i)
    {
        sum += arr[i];
    }

    avg = (double)sum / size;

    return avg;
}

@end

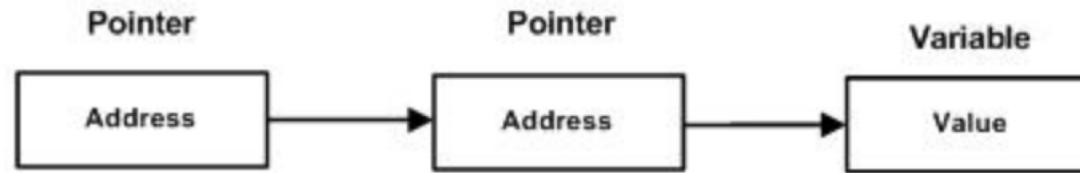
int main ()
{
    /* an int array with 5 elements */
    int balance[5] = {1000, 2, 3, 17, 50};
    double avg;

    SampleClass *sampleClass = [[SampleClass alloc] init];
    /* pass pointer to the array as an argument */
    avg = [sampleClass getAverage: balance ofSize: 5 ] ;

    /* output the returned value */
    NSLog(@"Average value is: %f\n", avg );

    return 0;
}
```

pointer de pointer



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

int main ()
{
    int var;
    int *ptr;
    int **pptr;

    var = 3000;

    /* take the address of var */
    ptr = &var;

    /* take the address of ptr using address of operator & */
    pptr = &ptr;

    /* take the value using pptr */
    NSLog(@"Value of var = %d\n", var );
    NSLog(@"Value available at *ptr = %d\n", *ptr );
    NSLog(@"Value available at **pptr = %d\n", **pptr);

    return 0;
}
```

```
Valeur de var = 3000
Valeur disponible à *ptr = 3000
Valeur disponible à **pptr = 3000
```

Strings en Objective-C

String représenté par **NSString** et **NSMutableString**

Syntaxe pour appeler les fonctions

typeDeRetour variableDeCapture = [objetDuDestinataire nomDuMéthodeArgument: valeurDuArgument]

typeDeRetour : Le type de variable auquel le méthode vas retourner

variableDeCapture : Le nom de la variable dans laquelle enregistrer le résultat de la méthode; il peut s'agir d'une nouvelle variable ou d'une variable existante, mais il doit correspondre au type de la méthode

objetDuDestinataire : L'objet sur lequel la méthode est appelée, qui doit exécuter le comportement attendu.

nomDuMéthodeArgument : Le nom de la méthode qui doit décrire son effet et qui est utilisé pour appeler son comportement. Le nom de la méthode peut contenir des spécificateurs d'argument.

valeurDuArgument : Les variables ou les valeurs doivent être transférées dans la méthode qui sont pertinentes pour son fonctionnement et dont la méthode à besoin pour être exécutée.

```
NSString *uppercase =[@"welcome!" uppercaseString];
```

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

```
NSString *welcome = @"Welcome!";
```

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

Strings en Objective-C

String représenté par **NSString** et **NSMutableString**

- **(BOOL)isEqualToString:(NSString *)aString**

Renvoie une valeur booléenne qui indique si une chaîne donnée est égale au récepteur à l'aide d'une comparaison littérale basée sur Unicode



```
NSString *myString1=@"Hello World";
NSString *myString2=@"Hello World";

if([myString1 isEqualToString:myString2]){

    NSLog(@"similar");
}

else{

    NSLog(@"dissimilar");
}

// it returns a boolean, a special int
NSLog(@"%@", [myString1 isEqualToString:myString2]);
```

- **(NSUInteger)length**

Renvoie le nombre de caractères Unicode dans le récepteur



```
/* total length of str3 after concatenation */
NSUInteger len = [str2 length];
NSLog(@"Length of Str2 : %lu", (unsigned long)len );

NSUInteger welcomeLength =[@"welcome." length];

NSLog(@"welcomeLength: %lu", welcomeLength);
```

```
// the strings are NOT exactly alike

if ([@"snoop dog" isEqualToString:@"Snoop Dog"]){
    NSLog(@"This will not print.");
} else {
    NSLog(@"Will the real Snoop Dog please stand up?");
}
```

Strings en Objective-C

String représenté par **NSString** et **NSMutableString**

```
NSString* myString = @"10:Username taken";  
  
if([myString hasPrefix:@"10"]){
    //display more elegant error message
}
```

```
BOOL doctor =[@"Doctor Who" hasPrefix:@"Doctor"];
```

```
NSLog(@"doctor: %d", doctor);
```

```
BOOL esquire =[@"John Smith, Esq." hasSuffix:@"Esq."];
```

```
NSLog(@"esquire: %d", esquire);
```

```
unichar myVal = [otherString characterAtIndex:5];
NSLog(@"The char is : %hu",myVal);
```

- **(BOOL)hasPrefix:(NSString *)aString**

Renvoie une valeur booléenne qui indique si une chaîne donnée correspond aux caractères de début du récepteur.

- **(BOOL)hasSuffix:(NSString *)aString**

Renvoie une valeur booléenne qui indique si une chaîne donnée correspond aux caractères de fin du destinataire

- **(unichar)characterAtIndex:(NSUInteger)index**

Renvoie le caractère à une position de tableau donnée

- **(id)initWithFormat:(NSString *)format ...**

Renvoie un objet NSString initialisé en utilisant une chaîne de format donnée en tant que modèle dans lequel les valeurs d'argument restantes sont remplacées

```
#import <Foundation/Foundation.h>  
  
int main ()
{
    NSString *str1 = @"Hello";
    NSString *str2 = @"World";
    NSString *str3;
    int len;  
  
NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];
/* InitWithFormat */
str3 = [NSString alloc] initWithFormat:@"%@ %@", str1, str2];
NSLog(@"%@", str3);
[pool drain];
  
return 0;
}
```

Strings en Objective-C

String représenté par **NSString** et **NSMutableString**

- **(NSInteger)integerValue**

Renvoie la valeur NSInteger du texte (de type NSNUmber, float, double etc.) du destinataire

- **(double)doubleValue**

Renvoie la valeur en virgule flottante du texte (de type NSNUmber, float, int etc.) du destinataire en double

- **(float)floatValue**

Renvoie la valeur en virgule flottante du texte (de type NSNUmber, double, int etc.) du destinataire en float

```
NSNumber *number = [NSNumber numberWithDouble:10.89555];
NSLog(@"%@", [number floatValue]); // To convert from NSNumber into float value
NSLog(@"%@", [number doubleValue]); // To convert from NSNulber into double value
```

```
NSNumber *numberInt = [NSNumber numberWithInt:8955];
NSLog(@"%@", [numberInt intValue]); // To convert from NSNumber into integer value
```

```
NSString *ageString = @"29";
NSUInteger age = [ageString integerValue];

age++; // birthday party!

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

Strings en Objective-C

String représenté par **NSString** et **NSMutableString**

- **(NSString *)uppercaseString**

Retourne une représentation en majuscule du récepteur

- **(NSString *)lowercaseString**

Retourne une représentation en minuscule du récepteur

- **(NSString *)capitalizedString**

Renvoie une représentation en majuscule du destinataire

```
NSString *welcome =[@"welcome." uppercaseString];  
  
NSLog(@"%@", welcome);  
  
NSString *welcome =[@"WELCOME!" lowercaseString];  
  
NSLog(@"%@", welcome);
```

```
27 int main(int argc, const char * argv[]) {  
28     @autoreleasepool {  
29  
30         NSString *str1 = @"Bonjour";  
31         NSString *str2 = @"Salut ça va";  
32  
33         NSString *str3;  
34         NSString *str4, *str5;  
35  
36         /* Converting into Upper Case String */  
37         str3 = [str2 uppercaseString];  
38         NSLog(@"The Uppercase String is : %@",str3);  
39  
40         /* Converting into Lower Case String */  
41         str4 = [str1 lowercaseString];  
42         NSLog(@"The Lowercase String is : %@",str4);  
43  
44         /* Converting into Upper Case String */  
45         str5 = [str2 capitalizedString];  
46         NSLog(@"The Capitalized String is : %@",str3);  
47  
48  
NSString *welcomeToFlatiron =[@"welcome to flatiron." capitalizedString];  
  
NSLog(@"%@", welcomeToFlatiron);
```

Strings en Objective-C

- **(NSRange)rangeOfString:(NSString *)aString**

Trouve et renvoie la plage de la première occurrence d'une chaîne donnée dans le récepteur.

```
NSString *string = @"hello bla bla";
if ([string rangeOfString:@"bla"].location == NSNotFound) {
    NSLog(@"string does not contain bla");
} else {
    NSLog(@"string contains bla!");
}
```

- **(NSString *)stringByAppendingFormat:(NSString *)format**

Renvoie une chaîne créée en ajoutant au récepteur une chaîne construite à partir d'une chaîne de format donnée et les arguments suivants

```
NSRange range = [name rangeOfString: @"Arnold"];
NSUInteger start = range.location;
NSUInteger end = start + range.length;
```

```
NSString *myString = @"some text: ";
myString = [myString stringByAppendingString:@" à quelle heure, Tu vas aller pour badminton = %d", 3];
NSLog(@"%@", myString);
```

```
NSString *aString =self.stringName;
NSString *resultString =[name stringByAppendingFormat:@"%@",aString];
```

Strings en Objective-C

- **(NSString *)stringByTrimmingCharactersInSet:(NSCharacterSet *)set**

Renvoie une nouvelle chaîne créée en supprimant des deux extrémités du récepteur les caractères contenus dans un jeu de caractères donné

```
NSString *string = @" this text has spaces before and after ";
NSString *trimmedString = [string stringByTrimmingCharactersInSet:
                           [NSCharacterSet whitespaceCharacterSet]];
```

- **(NSString *)substringFromIndex:(NSUInteger)anIndex**

Renvoie une nouvelle chaîne contenant les caractères du destinataire de celui d'un index donné à la fin.

```
NSString *myString = @"This is my String";
myString = [myString substringFromIndex:5];
```

Structures en Objective-C

Accéder le Structures

Structures comme les argument de fonctions

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

struct Books
{
    NSString *title;
    NSString *author;
    NSString *subject;
    int book_id;
};

int main( )
{
    struct Books Book1;      /* Declare Book1 of type Book */
    struct Books Book2;      /* Declare Book2 of type Book */

    /* book 1 specification */
    Book1.title = @"Objective-C Programming";
    Book1.author = @"Nuha Ali";
    Book1.subject = @"Objective-C Programming Tutorial";
    Book1.book_id = 6495407;

    /* book 2 specification */
    Book2.title = @"Telecom Billing";
    Book2.author = @"Zara Ali";
    Book2.subject = @"Telecom Billing Tutorial";
    Book2.book_id = 6495700;

    /* print Book1 info */
    NSLog(@"Book 1 title : %@", Book1.title);
    NSLog(@"Book 1 author : %@", Book1.author);
    NSLog(@"Book 1 subject : %@", Book1.subject);
    NSLog(@"Book 1 book_id : %d", Book1.book_id);

    /* print Book2 info */
    NSLog(@"Book 2 title : %@", Book2.title);
    NSLog(@"Book 2 author : %@", Book2.author);
    NSLog(@"Book 2 subject : %@", Book2.subject);
    NSLog(@"Book 2 book_id : %d", Book2.book_id);

    return 0;
}
```

```
Book title : Objective-C Programming
Book author : Nuha Ali
Book subject : Objective-C Programming Tu
Book book_id : 6495407
Book title : Telecom Billing
Book author : Zara Ali
Book subject : Telecom Billing Tutorial
Book book_id : 6495700
```

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

struct Books
{
    NSString *title;
    NSString *author;
    NSString *subject;
    int book_id;
};

@interface SampleClass:NSObject

/* function declaration */
- (void) printBook:( struct Books) book ;

@end

@implementation SampleClass

- (void) printBook:( struct Books) book
{
    NSLog(@"Book title : %@", book.title);
    NSLog(@"Book author : %@", book.author);
    NSLog(@"Book subject : %@", book.subject);
    NSLog(@"Book book_id : %d", book.book_id);
}

int main( )
{
    struct Books Book1;      /* Declare Book1 of type Book */
    struct Books Book2;      /* Declare Book2 of type Book */

    /* book 1 specification */
    Book1.title = @"Objective-C Programming";
    Book1.author = @"Nuha Ali";
    Book1.subject = @"Objective-C Programming Tutorial";
    Book1.book_id = 6495407;

    /* book 2 specification */
    Book2.title = @"Telecom Billing";
    Book2.author = @"Zara Ali";
    Book2.subject = @"Telecom Billing Tutorial";
    Book2.book_id = 6495700;

    SampleClass *sampleClass = [[SampleClass alloc] init];
    /* print Book1 info */
    [sampleClass printBook: Book1];

    /* Print Book2 info */
    [sampleClass printBook: Book2];

    return 0;
}
```

Structures en Objective-C

Pointers de Structure

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

struct Books
{
    NSString *title;
    NSString *author;
    NSString *subject;
    int book_id;
};

@interface SampleClass: NSObject

/* function declaration */
- (void) printBook:( struct Books *) book ;

@end

@implementation SampleClass

- (void) printBook:( struct Books *) book
{
    NSLog(@"Book title : %@", book->title);
    NSLog(@"Book author : %@", book->author);
    NSLog(@"Book subject : %@", book->subject);
    NSLog(@"Book book_id : %d\n", book->book_id);
}
@end

int main( )
{
    struct Books Book1;      /* Declare Book1 of type Book */
    struct Books Book2;      /* Declare Book2 of type Book */

    /* book 1 specification */
    Book1.title = @"Objective-C Programming";
    Book1.author = @"Nuha Ali";
    Book1.subject = @"Objective-C Programming Tutorial";
    Book1.book_id = 6495407;

    /* book 2 specification */
    Book2.title = @"Telecom Billing";
    Book2.author = @"Zara Ali";
    Book2.subject = @"Telecom Billing Tutorial";
    Book2.book_id = 6495700;

    SampleClass *sampleClass = [[SampleClass alloc] init];
    /* print Book1 info by passing address of Book1 */
    [sampleClass printBook:&Book1];

    /* print Book2 info by passing address of Book2 */
    [sampleClass printBook:&Book2];

    return 0;
}
```

```
Book title : Objective-C Programming
Book author : Nuha Ali
Book subject : Objective-C Programming T
Book book_id : 6495407
Book title : Telecom Billing
Book author : Zara Ali
Book subject : Telecom Billing Tutorial
Book book_id : 6495700
```

Typedef

-- **typedef**: peut être utilisé pour donner un nouveau nom à un type

```
typedef unsigned char BYTE;
```

-- l'identificateur BYTE peut être utilisé comme abréviation pour le type « `char unsigned` »

```
BYTE b1, b2;
```

-- Par convention, les lettres majuscules sont utilisées

mais nous pouvons aussi utiliser des minuscules, comme suit

```
typedef unsigned char byte;
```

-- Peut également utiliser **typedef** pour donner un nom au type de données défini par l'utilisateur
-- Pouvez utiliser le **typedef** avec structure pour définir un nouveau type de données
-- Peut utiliser ce type de données pour définir directement les variables de **structure**

Les diapositives sont faites à partir des matériaux à:
www.tutorialpoint.com

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

typedef struct Books
{
    NSString *title;
    NSString *author;
    NSString *subject;
    int book_id;
} Book;

int main( )
{
    Book book;
    book.title = @"Objective-C Programming";
    book.author = @"TutorialsPoint";
    book.subject = @"Programming tutorial";
    book.book_id = 100;
    NSLog(@"Book title : %@", book.title);
    NSLog(@"Book author : %@", book.author);
    NSLog(@"Book subject : %@", book.subject);
    NSLog(@"Book Id : %d\n", book.book_id);

    return 0;
}
```

```
...
Book title : Objective-C Programming
Book author : TutorialsPoint
Book subject : Programming tutorial
Book Id : 100
```

typedef vs #define

- Le **typedef** est limité à donner des noms symboliques aux types seulement alors que **#define** peut être utilisé pour définir un alias pour les valeurs aussi, comme vous pouvez définir 1 comme UN, etc
- L'interprétation **typedef** est exécutée par le compilateur où les instructions **#define** sont traitées par le préprocesseur

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

#define TRUE 1
#define FALSE 0

int main( )
{
    NSLog( @"Value of TRUE : %d\n", TRUE);
    NSLog( @"Value of FALSE : %d\n", FALSE);

    return 0;
}
```

Classes et Objets

-- Une définition de classe commence par le mot-clé **@interface** suivi du nom de l'interface (classe) et du corps de la classe, entouré d'une paire d'accolades bouclées

-- En Objective-C, toutes les classes sont dérivées de la classe de base appelée **NSObject**

-- C'est la superclasse de toutes les classes Objective-C

-- Il fournit des méthodes de base comme l'allocation de mémoire et l'initialisation

```
@interface Box: NSObject
{
    //Instance variables
    double length;    // Length of a box
    double breadth;   // Breadth of a box
}
@property(nonatomic, readwrite) double height; // Property
@end
```

```
Box box1 = [[Box alloc]init];    // Create box1 object of type Box
Box box2 = [[Box alloc]init];    // Create box2 object of type Box
```

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

@interface Box: NSObject
{
    double length;    // Length of a box
    double breadth;   // Breadth of a box
    double height;    // Height of a box
}
@property(nonatomic, readwrite) double height; // Property

-(double) volume;

@end

@implementation Box

@synthesize height;

-(id)init
{
    self = [super init];
    length = 1.0;
    breadth = 1.0;
    return self;
}

-(double) volume
{
    return length*breadth*height;
}

int main( )
{
    NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];
    Box *box1 = [[Box alloc]init];    // Create box1 object of type Box
    Box *box2 = [[Box alloc]init];    // Create box2 object of type Box

    double volume = 0.0;    // Store the volume of a box here

    // box 1 specification
    box1.height = 5.0;

    // box 2 specification
    box2.height = 10.0;

    // volume of box 1
    volume = [box1 volume];
    NSLog(@"Volume of Box1 : %f", volume);
    // volume of box 2
    volume = [box2 volume];
    NSLog(@"Volume of Box2 : %f", volume);
    [pool drain];
    return 0;
}
```

```
Volume of Box1 : 5.000000
Volume of Box2 : 10.000000
```

Heritage

Une classe dérivée hérite de toutes les méthodes et variables de la classe de base avec les exceptions suivantes:

- Les variables déclarées dans le fichier d'implémentation utilisant d'extensions ne sont pas accessibles.
- Les méthodes déclarées dans le fichier d'implémentation utilisant des extensions ne sont pas accessibles.
- Dans le cas où la classe héritée implémente la méthode dans la classe de base, la méthode dans la classe dérivée est exécutée.

Heritage

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

@interface Person : NSObject

{
    NSString *personName;
    NSInteger personAge;
}

- (id)initWithName:(NSString *)name andAge:(NSInteger)age;
- (void)print;
@end

@implementation Person

- (id)initWithName:(NSString *)name andAge:(NSInteger)age{
    personName = name;
    personAge = age;
    return self;
}

- (void)print{
    NSLog(@"Name: %@", personName);
    NSLog(@"Age: %ld", personAge);
}
@end

@interface Employee : Person

{
    NSString *employeeEducation;
}

- (id)initWithName:(NSString *)name andAge:(NSInteger)age
andEducation:(NSString *)education;
- (void)print;
@end
```

```
@implementation Employee

- (id)initWithName:(NSString *)name andAge:(NSInteger)age
andEducation: (NSString *)education
{
    personName = name;
    personAge = age;
    employeeEducation = education;
    return self;
}

- (void)print
{
    NSLog(@"Name: %@", personName);
    NSLog(@"Age: %ld", personAge);
    NSLog(@"Education: %@", employeeEducation);
}
@end

int main(int argc, const char * argv[])
{
    NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];
    NSLog(@"Base class Person Object");
    Person *person = [[Person alloc] initWithName:@"Raj" andAge:5];
    [person print];
    NSLog(@"Inherited Class Employee Object");
    Employee *employee = [[Employee alloc] initWithName:@"Raj"
    andAge:5 andEducation:@"MBA"];
    [employee print];
    [pool drain];
    return 0;
}
```

```
Base class Person Object
Name: Raj
Age: 5
Inherited Class Employee Object
Name: Raj
Age: 5
Education: MBA
```

Polymorphisme

Le polymorphisme

Objective-C signifie qu'un appel à une fonction provoquera l'exécution d'une fonction différente qui dépend sur le type d'objet qui appelle la fonction.

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

@interface Shape : NSObject
{
    CGFloat area;
}

- (void)printArea;
- (void)calculateArea;
@end

@implementation Shape
- (void)printArea{
    NSLog(@"The area is %f", area);
}

- (void)calculateArea{

}
@end

@interface Square : Shape
{
    CGFloat length;
}

- (id)initWithSide:(CGFloat)side;
- (void)calculateArea;
@end
```

```
@implementation Square
- (id)initWithSide:(CGFloat)side{
    length = side;
    return self;
}

- (void)calculateArea{
    area = length * length;
}

- (void)printArea{
    NSLog(@"The area of square is %f", area);
}

@implementation Rectangle
{
    CGFloat length;
    CGFloat breadth;
}

- (id)initWithLength:(CGFloat)rLength andBreadth:(CGFloat)rBreadth;
@end

@implementation Rectangle
- (id)initWithLength:(CGFloat)rLength andBreadth:(CGFloat)rBreadth{
    length = rLength;
    breadth = rBreadth;
    return self;
}

- (void)calculateArea{
    area = length * breadth;
}

int main(int argc, const char * argv[])
{
    NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];
    Shape *square = [[Square alloc] initWithSide:10.0];
    [square calculateArea];
    [square printArea];
    Shape *rect = [[Rectangle alloc]
        initWithLength:10.0 andBreadth:5.0];
    [rect calculateArea];
    [rect printArea];
    [pool drain];
    return 0;
}
```

Préprocesseur

#define	Substitute une macro de préprocesseur
#include	Insère un en-tête particulier d'un autre fichier
#undef	Undefines une macro de préprocesseur
#ifdef	Renvoie true si cette macro est définie
#ifndef	Reenvoie true si cette macro n'est pas définie
#if	Teste si une condition est vraie ou pas pendant compilation
#else	L'alternative pour #if
#elif	#else un #if dans une déclaration
#endif	Termine le préprocesseur conditionnel
#error	Imprime un message d'erreur sur stderr
#pragma	Emet des commandes spéciales au compilateur en utilisant une méthode standardisée

Préprocesseur

-- remplacer les instances de MAX_ARRAY_LENGTH par 20

```
#define MAX_ARRAY_LENGTH 20
```

-- Obtenir foundation.h de Foundation Framework

-- Récupérer myheader.h du répertoire local et d'ajouter le contenu

```
#import <Foundation/Foundation.h>
#include "myheader.h"
```

-- à ne pas définir FILE_SIZE existant et de le définir comme 42

```
#undef FILE_SIZE
#define FILE_SIZE 42
```

-- à ne définir MESSAGE que si MESSAGE n'est pas déjà défini.

```
#ifndef MESSAGE
#define MESSAGE "You wish!"
#endif
```

-- à faire le processus les instructions jointes si DEBUG est défini

```
#ifdef DEBUG
    /* Your debugging statements here */
#endif
```

Macros prédéfinies

<code>_DATE_</code>	Affiche date actuelle en tant que caractère littéral dans le format « MMM DD YYYY »
<code>_TIME_</code>	L'heure actuelle en tant que caractère littéral au format "HH: MM: SS"
<code>_FILE_</code>	Cela contient le nom de fichier actuel en tant que string littérale
<code>_LINE_</code>	Ceci contient le numéro de ligne actuel en tant que constante décimale
<code>_STDC_</code>	Défini comme 1 lorsque le compilateur est conforme à la norme ANSI

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

int main()
{
    NSLog(@"File :%s\n", __FILE__ );
    NSLog(@"Date :%s\n", __DATE__ );
    NSLog(@"Time :%s\n", __TIME__ );
    NSLog(@"Line :%d\n", __LINE__ );
    NSLog(@"ANSI :%d\n", __STDC__ );

    return 0;
}
```

```
File :main.m
Date :Sep 14 2013
Time :04:46:14
Line :8
ANSI :1
```

Opérateurs de préprocesseurs

Macro Continuation : L'opérateur de macro continuation est utilisé pour continuer une macro trop longue pour une seule ligne.

```
#define message_for(a, b) \
    NSLog(@"#a " and "#b ": We love you!\n")
```

```
#define NINE (3 \
+ 3)
```

It's Six

```
#include <stdio.h>

#define COMPARE(x,y) ((x)<(y)?-1:((x)==(y)?0:1))

int main()
{
    double a = 1.5;
    int n = 2;
    float z = 1.1f;
    printf("\n compare %lf (double) with %f (float): %d", a , z, COMPARE(a,z));
    printf("\n compare %lf (double) with %d (int): %d", a , n, COMPARE(a,n));
    printf("\n compare %d (int) with %f (float): %d", n , z, COMPARE(n,z));
    printf("\n compare \" n * a\" = %lf \n"
        " with \"n+1\" = to %d (double): %d",
        n*a , n+1, COMPARE(n*a,n+1));
    printf("\n");
}
```

Opérateurs de préprocesseurs

Stringize/Signe Numérique (#):

- lorsqu'il est utilisé dans une définition de macro, convertit un paramètre macro en une constante de chaîne
- Cet opérateur ne peut être utilisé que dans une macro qui à un argument ou une liste de paramètres spécifiés

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

#define message_for(a, b) \
    NSLog(@"%@", #b ": We love you!\n")

int main(void)
{
    message_for(Carole, Debra);
    return 0;
}
```

```
2013-09-14 05:46:14.859 demo[20683] Carole and Debra: We love you!
```

Opérateurs de préprocesseurs

L'opérateur de collage de jetons (##):

- Dans une définition de macro, il combine deux arguments.
- Il permet d'associer deux jetons séparés dans la définition de macro en un seul jeton

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

#define tokenpaster(n) NSLog (@"token" #n " = %d", token##n)

int main(void)
{
    int token34 = 40;

    tokenpaster(34);
    return 0;
}
```

```
2013-09-14 05:48:14.859 demo[20683] token34 = 40
```

Opérateurs de préprocesseurs

L'opérateur défini ():

- L'opérateur **defined** par le préprocesseur est utilisé dans les expressions constantes
 - Pour déterminer si un identificateur est défini par **#define**
- Si l'identificateur spécifié est défini, la valeur est true (non-zéro)
- Si le symbole n'est pas défini, la valeur est fausse (zéro)

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

#if !defined (MESSAGE)
    #define MESSAGE "You wish!"
#endif

int main(void)
{
    NSLog(@"%@", @"Here is the message: %s\n", MESSAGE);
    return 0;
}
```

```
2013-09-14 05:48:19.859 demo[20683] Here is the message: You wish!
```

Opérateurs de préproesseurs

- Les macros avec des arguments doivent être définies par la directive **#define** avant l'utilisation
- La liste des arguments est entre parenthèses et doit immédiatement suivre le nom de la macro
- Les espaces ne sont pas autorisés entre le nom de la macro et les parenthèses ouvertes.

```
int square(int x) {  
    return x * x;  
}
```



```
#define square(x) ((x) * (x))
```

```
#import <Foundation/Foundation.h>  
  
#define MAX(x,y) ((x) > (y) ? (x) : (y))  
  
int main(void)  
{  
    NSLog(@"%@", @"Max between 20 and 10 is %d\n", MAX(10, 20));  
    return 0;  
}
```

```
2013-09-14 05:52:15.859 demo[20683] Max between 20 and 10 is 20
```

Type casting

Promotion de Integer

-- Le division de deux valeur Integer

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

int main()
{
    int sum = 17, count = 5;
    CGFloat mean;

    mean = (CGFloat) sum / count;
    NSLog(@"Value of mean : %f\n", mean );

    return 0;
}
```

Value of mean : 3.400000

l'opérateur de casting à préséance sur la division,

donc la valeur de la «sum» est d'abord convertie en double et finalement elle est divisée par le «count», donnant une **double**

- Le compilateur exécute d'abord la promotion des integer
- Si les opérandes ont encore des types différents alors ils sont convertis au type qui apparaît le plus haut dans la hiérarchie

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

int main()
{
    int i = 17;
    char c = 'c'; /* ascii value is 99 */
    int sum;

    sum = i + c;
    NSLog(@"Value of sum : %d\n", sum );

    return 0;
}
```

Value of sum : 116

- d'abord « c » est converti en integer

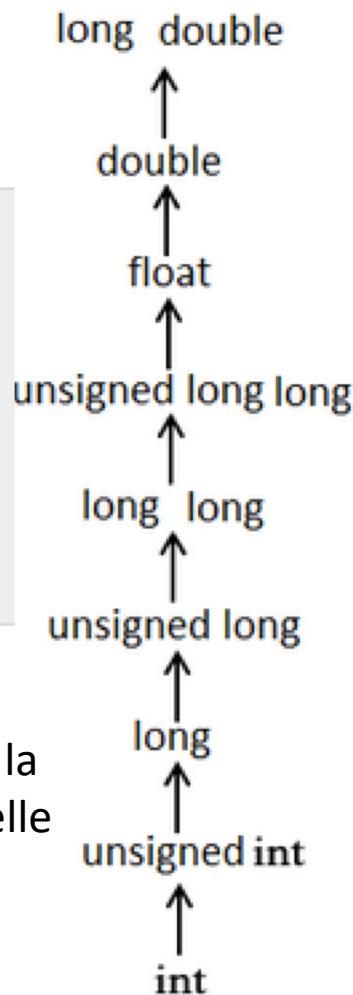
```
#import <Foundation/Foundation.h>

int main()
{
    int i = 17;
    char c = 'c'; /* ascii value is 99 */
    CGFloat sum;

    sum = i + c;
    NSLog(@"Value of sum : %f\n", sum );
    return 0;
}
```

Value of sum : 116.000000

- la valeur finale est float, donc la conversion arithmétique habituelle s'applique et le compilateur convertir « i » et « c » en float



Catégories

- Pour ajouter une méthode à une classe existante, peut-être, d'ajouter des fonctionnalités pour rendre plus facile de faire quelque chose dans notre propre application
- Une catégorie peut être déclarée pour n'importe quelle classe, même si vous n'avez pas le code original
- Toutes les méthodes que vous déclarez dans une catégorie seront disponibles pour toutes les instances de la classe d'origine, ainsi que pour toutes les sous-classes de la classe d'origine
- Lors de l'exécution, il n'y a aucune différence entre une méthode ajoutée par une catégorie et une méthode implémentée par la classe d'origine

```
@interface ClassName (CategoryName)  
@end
```

Catégories

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

@interface NSString(MyAdditions)
+(NSString *)getCopyRightString;
@end

@implementation NSString(MyAdditions)
+(NSString *)getCopyRightString{
    return @"Copyright TutorialsPoint.com 2013";
}
@end

int main(int argc, const char * argv[])
{
    NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];
    NSString *copyrightString = [NSString getCopyRightString];
    NSLog(@"Accessing Category: %@",copyrightString);
    [pool drain];
    return 0;
}
```

-- vous devrez importer le fichier d'en-tête de catégorie dans n'importe quel fichier de code source où vous souhaitez utiliser les méthodes supplémentaires

Un peu plus des exemples du Catégories

La méthode **reverseString** ajoute la possibilité à tous les objets **NSString** d'inverser les caractères de la chaîne.

```
1 @interface NSString (reverse)
2 -(NSString *) reverseString;
3 @end
```

Comme pour la déclaration **@interface**, la section **@implementation** ne change que dans la mesure où le nom de la catégorie est ajouté.

```
1 @implementation NSString (reverse)
2 -(NSString *) reverseString
3 {
4     NSMutableString *reversedStr;
5     int len = [self length];
6
7     // Auto released string
8     reversedStr = [NSMutableString stringWithCapacity:len];
9
10    // Probably woefully inefficient...
11    while (len > 0)
12        [reversedStr appendString:
13         [NSString stringWithFormat:@"%C", [self characterAtIndex:--len]]];
14
15    return reversedStr;
16 }
17
18 @end
```

Comment utiliser la catégorie

- **NSString+Reverse.h**
- **NSString+Reverse.m**

convention de nommage

ositives sont faites à partir des matériaux à:
www.tutorialpoint.com

```
1 #import <Foundation/Foundation.h>
2 #import "NSString+Reverse.h"
3
4 int main (int argc, const char * argv[])
5 {
6     NSAutoreleasePool *pool = [[NSAutoreleasePool alloc] init];
7     NSString *str = [NSString stringWithFormat:@"Fubar"];
8     NSString *rev;
9
10    NSLog(@"String: %@", str);
11    rev = [str reverseString];
12    NSLog(@"Reversed: %@", rev);
13
14    [pool drain];
15    return 0;
16 }
```

Utilisations du Catégories

- Pour diviser la fonctionnalité d'une classe en plusieurs fichiers
- Vous travaillez avec d'autres personnes sur les mêmes classe et vous avez été chargé de gérer certaines parties de cette classe
- Vous devez ajouter des méthodes à une classe à laquelle vous n'avez pas accès, comme les classes Foundation (`NSString`, `NSArray`)

Quelque petite notes

Une méthode de catégorie peut remplacer une méthode existante, mais vous perdrez l'accès à l'original en dehors de cette classe

Par conséquent, si vous **over ride** cette fonction, assurez-vous de dupliquer les fonctionnalités existantes ou utilisez un appel par super

Si une méthode est définie dans plusieurs catégories d'une même classe, la version invoquée on sais pas !!

Extensions

- Les méthodes déclarées par une extension de classe **pour qui, on à des source code**
- Pas possible à déclarer une **extension** de classe sur une classe framework, e.g. classe **Cocoa** ou **Cocoa Touch** comme **NSString**
- Les **extensions sont en fait des catégories sans nom de catégorie**. On parle souvent de catégories anonymes
- Une extension **ajoute des méthodes privées et des variables privées** qui ne sont spécifiques qu' à la classe
- Toute méthode ou variable déclarée dans les extensions n'est pas accessible même aux classes héritées

Extensions

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

@interface SampleClass : NSObject
{
    NSString *name;
}

- (void)setInternalID;
- (NSString *)getExternalID;

@end

@interface SampleClass()
{
    NSString *internalID;
}

@end

@implementation SampleClass

- (void)setInternalID{
    internalID = [NSString stringWithFormat:@"UNIQUEINTERNALKEY%dUNIQUEINTERNALKEY",arc4random()%100];
}

- (NSString *)getExternalID{
    return [internalID stringByReplacingOccurrencesOfString:@"UNIQUEINTERNALKEY" withString:@""];
}

@end

int main(int argc, const char * argv[])
{
    NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];
    SampleClass *sampleClass = [[SampleClass alloc] init];
    [sampleClass setInternalID];
    NSLog(@"ExternalID: %@",[sampleClass getExternalID]);
    [pool drain];
    return 0;
}
```

- Nous pouvons voir que **InternalID** n'est pas renvoyé directement
- Nous supprimons ici l'**UNIQUEINTERNALKEY** et ne rendons disponible que la valeur restante à la méthode **getExternalID**

Différence entre Extensions et Catégories

Lorsque vous souhaitez ajouter des fonctionnalités supplémentaires à une classe sans héritage, vous utilisez simplement la catégorie pour cette classe

Utiliser la catégorie lorsque vous devez diviser le même code de classe dans un fichier source différent selon les différentes fonctionnalités

Quand vous voulez rendre le comportement d'une propriété privée, ou ajouter les fonction privée, vous utilisez l'extension de classe

Extension lorsque vous avez juste besoin d'ajouter quelques méthodes requises à la classe existante en dehors du fichier d'interface principale

également lorsque vous devez modifier une variable, déclarée publiquement dans une classe

Protocoles

- Les protocoles déclarent les méthodes qui devraient être utilisées dans une situation particulière
- Les protocoles sont implémentés dans les classes conformes au protocole

La syntaxe de Protocole :

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

La syntaxe de la classe utilise Protocole:

```
@interface MyClass : NSObject <MyProtocol>  
...  
@end
```

Toute instance de **MyClass** répondra aux méthodes déclarées dans l'interface de **MyClass**, mais aussi aux méthodes de **MyProtocol**

Il n'est pas nécessaire de **redéclarer** les méthodes de protocole dans l'interface de classe - l'adoption du protocole est suffisante.

Si vous avez besoin d'une classe pour **adopter plusieurs protocoles**, vous pouvez les spécifier comme liste séparée par des virgules

Protocoles

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

@protocol PrintProtocolDelegate
- (void)processCompleted;
@end

@interface PrintClass :NSObject
{
    id delegate;
}

- (void) printDetails;
- (void) setDelegate:(id)newDelegate;
@end

@implementation PrintClass
- (void)printDetails{
    NSLog(@"Printing Details");
    [delegate processCompleted];
}
- (void) setDelegate:(id)newDelegate{
    delegate = newDelegate;
}
@end
```

```
@interface SampleClass: NSObject<PrintProtocolDelegate>
- (void)startAction;
@end

@implementation SampleClass
- (void)startAction{
    PrintClass *printClass = [[PrintClass alloc] init];
    [printClass setDelegate:self];
    [printClass printDetails];
}

-(void)processCompleted{
    NSLog(@"Printing Process Completed");
}

int main(int argc, const char * argv[])
{
    NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];
    SampleClass *sampleClass = [[SampleClass alloc] init];
    [sampleClass startAction];
    [pool drain];
    return 0;
}
```

Properties

- Les propriétés sont introduites dans Objective-C pour garantir l'accès à la variable de la classe en dehors de la classe
- C'est seulement possible avec les propriétés que nous pouvons accéder aux variables d'instance de la classe. En fait, des méthodes getter et setter internes sont créées pour les propriétés
- Par exemple, supposons que nous ayons une propriété **@property (nonatomic, readonly) BOOL isDone**. Sous le capot, il y a des setters et getters créés comme indiqué ci-dessous.

```
-(void)setIsDone(BOOL)isDone;  
-(BOOL)isDone;
```

Properties

```
1 @interface RootViewController : UITableViewController {  
2     CGFloat mFontSize;  
3     NSArray* mFontNames;  
4 }  
5  
6 @property (nonatomic, assign) CGFloat fontSize;  
7 @property (nonatomic, retain) NSArray* fontNames;  
8  
9 @end
```

Le compilateur peut générer automatiquement (synthesis) les getters et setter pour vous.

Vous lui donnez les instruction pour le faire en utilisant la directive **@synthesize**

Les attributs est une liste séparée par des virgules. Les attributs appartiennent à quelques catégories:

Writability

readwrite: la **property** peut être écrite et lue. C'est la valeur par défaut

readonly: la **property** peut seulement être lue

Sémantique Setter

assign: la valeur entrante/incoming sera assignée à la **property**. Utilisez-le pour des types de données simples (p. ex. int, double)

retain: la valeur entrante sera conservée. Si la valeur est un objet Objective-C, c'est le plus courant.

copy: la valeur entrante sera copiée. Ceci est utilisé s'il y a un risque que l'objet entrant change (p. ex. NSMutableString)

Les diapositives sont faites à partir des matériaux à:
www.tutorialpoint.com

Atomicité

nonatomic: sont atomiques par défaut. Utilisez-le pour éviter le verrouillage (Locking)

Gérer Log

-- Normalement on écrit

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

int main()
{
    NSLog(@"Hello, World! \n");
    return 0;
}
```

Arrêter Log dans appli actuel

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

#if DEBUG == 0
#define DebugLog(...)
#elif DEBUG == 1
#define DebugLog(...) NSLog(__VA_ARGS__)
#endif

int main()
{
    DebugLog(@"Debug log, our custom addition gets \
printed during debug only" );
    NSLog(@"NSLog gets printed always" );
    return 0;
}
```

→ Debug log, our custom addition gets printed during debug only
NSLog gets printed always

→ NSLog gets printed always

Gérer le Erreur

```
SString *domain = @"com.MyCompany.MyApplication.ErrorDomain";
SString *desc = NSLocalizedString(@"Unable to complete the process", @"");
SDictionary *userInfo = @{
    NSLocalizedDescriptionKey : desc
};
SError *error = [NSError errorWithDomain:domain code:-101 userInfo:userInfo];
```

```
#import <Foundation/Foundation.h>
@interface SampleClass: NSObject
-(NSString *) getEmployeeNameForID:(int) id withError:(NSError **)errorPtr;
@end
@implementation SampleClass
-(NSString *) getEmployeeNameForID:(int) id withError:(NSError **)errorPtr{
    if(id == 1)
    {
        return @"Employee Test Name";
    }
    else
    {
        NSString *domain = @"com.MyCompany.MyApplication.ErrorDomain";
        NSString *desc =@"Unable to complete the process";
        NSDictionary *userInfo = [NSDictionary alloc]
        initWithObjectsAndKeys:desc,
        @NSLocalizedStringDescriptionKey,NULL];
        *errorPtr = [NSError errorWithDomain:domain code:-101
        userInfo:userInfo];
        return @"";
    }
}
int main()
{
    NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];
    SampleClass *sampleClass = [[SampleClass alloc]init];
    NSError *error = nil;
    NSString *name1 = [sampleClass getEmployeeNameForID:1 withError:&error];
    if(error)
    {
        NSLog(@"Error finding Name1: %@",error);
    }
    else
    {
        NSLog(@"Name1: %@",name1);
    }
    error = nil;
    NSString *name2 = [sampleClass getEmployeeNameForID:2 withError:&error];
    if(error)
    {
        NSLog(@"Error finding Name2: %@",error);
    }
    else
    {
        NSLog(@"Name2: %@",name2);
    }
    [pool drain];
    return 0;
}
```

```
Name1: Employee Test Name
Error finding Name2: Unable to complete the process
```

Encapsulation des données

Objective-C sont composés des deux éléments fondamentaux :

- **Déclarations de programme (code):** La partie d'un programme qui exécute des actions et elles sont appelées méthodes
- **Données du programme:** Les données sont les informations du programme qui sont affectées par les fonctions du programme

- Lie les données et les fonctions qui manipulent les données
- Protège à la fois des interférences extérieures et de l'usage abusif
- C'est un dissimulation des données
- C'est un mécanisme de regroupement des données et des fonctions
- C'est un mécanisme qui expose uniquement les interfaces et cache les détails de l'implémentation à l'utilisateur

Encapsulation des données

- Objective-C prend en charge les propriétés d'encapsulation et de dissimulation des données
- Grâce à la création de types définis par l'utilisateur, appelés classes

```
@interface Adder : NSObject
{
    NSInteger total;
}

- (id)initWithInitialNumber:(NSInteger)initialNumber;
- (void)addNumber:(NSInteger)newNumber;
- (NSInteger)getTotal;

@end
```

- La variable **total** est privée et nous ne pouvons pas accéder à l'extérieur de la classe
- La variable **total** est privée et nous ne pouvons pas accéder à l'extérieur de la classe

- Peuvent être accédés que par d'autres membres de la classe **Adder**
- C'est une façon d'obtenir l'encapsulation
- Les méthodes contenues dans «**interface**» sont accessibles en public