OBJECTIVE-C STRUCTURES

Objective-C arrays allow you to define type of variables that can hold several data items of the same kind but **structure** is another user-defined data type available in Objective-C programming which allows you to combine data items of different kinds.

Structures are used to represent a record, Suppose you want to keep track of your books in a library. You might want to track the following attributes about each book:

- Title

- Author

- Subject

- Book ID

Defining a Structure

To define a structure, you must use the **struct** statement. The struct statement defines a new data type, with more than one member for your program. The format of the struct statement is this:

```
struct [structure tag]
{
   member definition;
   member definition;
   ...
   member definition;
} [one or more structure variables];
```

The **structure tag** is optional and each member definition is a normal variable definition, such as int i; or float f; or any other valid variable definition. At the end of the structure's definition, before the final semicolon, you can specify one or more structure variables but it is optional. Here is the way you would declare the Book structure:

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

Accessing Structure Members

To access any member of a structure, we use the **member access operator** .. The member access operator is coded as a period between the structure variable name and the structure member that we wish to access. You would use **struct** keyword to define variables of structure type. Following is the example to explain usage of structure:

```
#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 */
```

```objc
   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 : %@\n", Book1.title);
   NSLog(@"Book 1 author : %@\n", Book1.author);
   NSLog(@"Book 1 subject : %@\n", Book1.subject);
   NSLog(@"Book 1 book_id : %d\n", Book1.book_id);

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

   return 0;
}
```

When the above code is compiled and executed, it produces the following result:

```
2013-09-14 04:20:07.947 demo[20591] Book 1 title : Objective-C Programming
2013-09-14 04:20:07.947 demo[20591] Book 1 author : Nuha Ali
2013-09-14 04:20:07.947 demo[20591] Book 1 subject : Objective-C Programming Tutorial
2013-09-14 04:20:07.947 demo[20591] Book 1 book_id : 6495407
2013-09-14 04:20:07.947 demo[20591] Book 2 title : Telecom Billing
2013-09-14 04:20:07.947 demo[20591] Book 2 author : Zara Ali
2013-09-14 04:20:07.947 demo[20591] Book 2 subject : Telecom Billing Tutorial
2013-09-14 04:20:07.947 demo[20591] Book 2 book_id : 6495700
```

Structures as Function Arguments

You can pass a structure as a function argument in very similar way as you pass any other variable or pointer. You would access structure variables in the similar way as you have accessed in the above example:

```objc
#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 : %@\n", book.title);
```

```
    NSLog(@"Book author : %@\n", book.author);
    NSLog(@"Book subject : %@\n", 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 */
    [sampleClass printBook: Book1];

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

    return 0;
}
```

When the above code is compiled and executed, it produces the following result:

```
2013-09-14 04:34:45.725 demo[8060] Book title : Objective-C Programming
2013-09-14 04:34:45.725 demo[8060] Book author : Nuha Ali
2013-09-14 04:34:45.725 demo[8060] Book subject : Objective-C Programming Tutorial
2013-09-14 04:34:45.725 demo[8060] Book book_id : 6495407
2013-09-14 04:34:45.725 demo[8060] Book title : Telecom Billing
2013-09-14 04:34:45.725 demo[8060] Book author : Zara Ali
2013-09-14 04:34:45.725 demo[8060] Book subject : Telecom Billing Tutorial
2013-09-14 04:34:45.725 demo[8060] Book book_id : 6495700
```

Pointers to Structures

You can define pointers to structures in very similar way as you define pointer to any other variable as follows:

```
struct Books *struct_pointer;
```

Now, you can store the address of a structure variable in the above-defined pointer variable. To find the address of a structure variable, place the & operator before the structure's name as follows:

```
struct_pointer = &Book1;
```

To access the members of a structure using a pointer to that structure, you must use the -> operator as follows:

```
struct_pointer->title;
```

Let us re-write above example using structure pointer, hope this will be easy for you to understand the concept:

```objc
#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 : %@\n", book->title);
    NSLog(@"Book author : %@\n", book->author);
    NSLog(@"Book subject : %@\n", 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;
}
```

When the above code is compiled and executed, it produces the following result:

```
2013-09-14 04:38:13.942 demo[20745] Book title : Objective-C Programming
2013-09-14 04:38:13.942 demo[20745] Book author : Nuha Ali
2013-09-14 04:38:13.942 demo[20745] Book subject : Objective-C Programming Tutorial
2013-09-14 04:38:13.942 demo[20745] Book book_id : 6495407
2013-09-14 04:38:13.942 demo[20745] Book title : Telecom Billing
2013-09-14 04:38:13.942 demo[20745] Book author : Zara Ali
2013-09-14 04:38:13.942 demo[20745] Book subject : Telecom Billing Tutorial
2013-09-14 04:38:13.942 demo[20745] Book book_id : 6495700
```

Bit Fields

Bit Fields allow the packing of data in a structure. This is especially useful when memory or data storage is at a premium. Typical examples:

- Packing several objects into a machine word. e.g. 1 bit flags can be compacted.

- Reading external file formats -- non-standard file formats could be read in. E.g. 9 bit integers.

Objective-C allows us do this in a structure definition by putting :bit length after the variable. For example:

```
struct packed_struct {
    unsigned int f1:1;
    unsigned int f2:1;
    unsigned int f3:1;
    unsigned int f4:1;
    unsigned int type:4;
    unsigned int my_int:9;
} pack;
```

Here, the packed_struct contains 6 members: Four 1 bit flags f1..f3, a 4 bit type and a 9 bit my_int.

Objective-C automatically packs the above bit fields as compactly as possible, provided that the maximum length of the field is less than or equal to the integer word length of the computer. If this is not the case, then some compilers may allow memory overlap for the fields whilst other would store the next field in the next word

Loading [MathJax]/jax/output/HTML-CSS/jax.js