Principles of Programming Languages

Assignment -2

Group:

Sagar Gupta	2014A7PS030H
Ravi Teja GVS	2014A7PS196H
Anirud Samala	2014A7PS027H
Dhawal Agrawal	2014A7PS112H

Language

- The language that was chosen for this assignment, is Actionscript 3.0

- It is an OOP, developed by Adobe, to support the Programming end, of the Adobe Flash Player Platform.

- Coupled with the web, it has been used to create powerful content-driven online tools, such as Video players, Games and Animations.

- It is also used with Adobe AIR system, for the development of desktop and mobile applications.

- Also, the language is open Source. We hence, also have a host of third party open-source compilers and VMs.

Data Types

- Boolean
- int
- Number
- Sreinf
- uint
- void
- Object
- A *Primitive* is one of the following : Boolean, int, number, String and uint. The rest are complex.
- All datatypes in Actionscript 3.0 whether they are primitive or complex, are objects.
- Working with primitive values in ActionScript 3.0 is usually faster than working with complex values, because Actionscript stores primitive values, that make memory and speed optimisations possible.

Boolean

- Comprises of two entities: true and false.

- The default value of a boolean variable that has been declared, but not initialised, if *false.*

int

- The int-datatype is stored internally as a 32-bit number.

- The default value for int is 0.

- The range of values that it includes, is -2,147,483,648 (-2^{31}) to 2,147,483,647 ($2^{31} - 1$), inclusive

Number

- The number datatype can represent, integers, unsigned integers and floating point numbers.

- If we don't have a decimal point within the declaration of the number datatype, internally, the compiler treats it as an *int* datatype.

- Its uses the 64 - bit double precision format as specified by IEEE Standard.

- MAX value : 1.7976931348623 1e+308

- MIN value: 4.9406564584124 67e-324

String

- Strings in ActionScript are *immutable,* Java.
- Hence, any operation on a string gives us back, an *instance of the string.*
- The default value of a string is *null.*
- It is stored as a string of 16-bit characters, as dictated by the Unicode norm of character encoding (UTF-16).

uint

- Is the *unsigned integer.*
- It ranges from 0 to 4,294,967,295.
- The default value is 0.

void

- The void datatype consists of only one value, *undefined.*

- The default value of Object instances is *null*, not to be confused with *void.*

- In fact, void can only be used as a return type annotation.

Object datatype

- The object datatype, stems from the object Class, in Actionscript.

- The object class, serves as the base class, for all class definitions in ActionScript.

- The default value for instances of the *Object* class, is *null.*

- Supports no-type-annotation:

  var x:*; —> Tells that x can be any type, until assigned.

- Such untyped variables hold the value *undefined.*

Complex Datatypes

- We also have support for complex datatypes in ActionScript. They are listed below.
- Array
- Date
- Error
- BitMap
- MovieClip
- Shape
- SimpleButton
- Sprite
- Video
- Function
- Object
- RegExp

Complex Datatypes (contd.)

- vector
- XML
- XMLList
- TextField
- ByteArray
- Dictionary

Run Time memory model

- Since all the variables in ActionScript are treated as objects, they are treated in the conventional sense, that objects are generally treated as.

- When we define any variable using its constructor:-
  var mySprite:Sprite = new Sprite();
  The mySprite variable, resides on the Stack, and references the location of the actual object on the *heap,* where the actual object definition resides.

- When we declare any variable with the following format:-
  var x:*;
  We postpone its datatype assignment until runtime, where we know what data type x eventually turns out to be. (denoted by the '*' symbol.)

Run Time memory model

- When we declare any object (only primitive datatypes), through the *equality* operator, that object is pushed onto the stack, and is popped out, once it goes out of scope.

- Also, ActionScript follows Dynamic Scoping. Hence, if the instance of the variable is not found, its instance is searched for in all its parents scopes/function till it is found.

- If it is not found there as well, then the search is continued in the global space.

Run Time memory model

- There are three types of spaces in the ActionScript memory model.
- Global: stores Global and static variables.
- Stack: stores local and function parameters (only primitive datatypes)
- Heap: stored dynamically allocated objects.
- The memory-address binding is done during the *run-time.* The binding is dynamic as well. (in view of declarations such as var x:*)

Parameter Passing

- In Actionscript 3.0, all arguments are passed by reference, because all values are stored as objects, and all objects are passed by reference.

- But, objects that belong to the primitive datatypes, have special operators that make them behave as if they were passed by value.

- Turn to the next slide, for an example on the same.

Parameter Passing (example)

```
function passPrimitives(xParam:int, yParam:int):void
{
    xParam++;
    yParam++;
    trace(xParam, yParam);
}

var xValue:int = 10;
var yValue:int = 15;
trace(xValue, yValue);// 10 15
passPrimitives(xValue, yValue); // 11 16
trace(xValue, yValue);// 10 15
```

Parameter Passing (contd.)

- In the previous slide, the function passPrimitives, takes two arguments, xParam and yParam.

- When the function is called, with xValue and yValue as parameters, then *the references of* xValue and yValue are stored in xParam and yParam.

- But any subsequent change *within* the function is done to a *copy* of the parameters within the function, and the same is returned.

Parameter Passing(contd.)

- A similar behaviour is observed in the String datatype as well.

- Though strings are passed by reference, since the strings in Actionscript are immutable, the changes are done to *a copy* of the string within the function (both, the string library function as well as a user defined function), and the *copy is returned.*

- A similar behaviour is observed in Java.

Garbage Collection

- The Actionscript language follows the *mark and sweep* garbage collection algorithm.

- Basically, the root set is scanned, and all the references that can be reached thorough it (recursively) are marked.

- Once the program is completed, we scan through the heap, and garbage collect all those references, whose BIT is not set (or not marked).

Garbage Collection (contd.)

- Internally, the garbage collection is done via a call to: flash.system.System.gc()

- This does a mark OR a sweep on a given object. (but not both, on the same call)

- So, inorder to have the effect of releasing the memory *back* to the OS, it is called twice in a row.

- The first call, is to mark any dereferenced objects and sweep away old marks.

- The second, is to sweep away marks from the first call.

- So, to tell the GC to remove the object, we simply set its value to *null*.

Special features

- It allows for the declaration of a variable *with no datatype, until initialised.* For e.g.:
  var x:*;

- All variables in ActionScript are treated as objects. Even primitive datatypes, are ‘wrapped’ in an object wrapper, to achieve this consistency.

- The availability of a (…(rest)) parameter, that accepts a variable number of arguments, that can be stored in an array. (not there in C) Its datatype definition is postponed until runtime.

Special Features (contd.)

- To remove an object (without memory leaks) by the GC, instead of deleting it, the object reference, is simply set to *null.*
- **Note:** Setting a reference to a display object to null does not ensure that the object is frozen. The object continues consume CPU cycles until it is garbage collected. We need to make sure that we properly deactivate your object before setting its reference to null. For e.g.: if we have an event listener object, we need to first call the removeEventListeners() method, before setting its reference to *null.*

Thank You!