Department Of Computer Engineering

CS 315 Programming Languages

Homework 1

Berdan Akyürek 21600904 Section 2

1. Sample Codes and Results

For each example program in every language, program creates an associative array of my friends and their student id numbers, where key is the name and value is id number. Later, program adds a new friend, removes a friend and modifies the number of an existing friend with printing the current array in each step. Later, program performs search operations for friend names and id numbers and prints the search results (true if found, false if not found). Lastly, program iterates over each friend and prints his/her name and id by calling a user defined function named foo.

1.1 Dart

1.1.1 Code

```
import 'dart:convert';

foo(var k, var v)
{
   print("$k: $v");
}

void main()
{
   var myArray = {"Berdan": 21600904, "Omer": 21100999, "Eylul":
21602019, "Serhat": 21704354, "Gunes": 21602271, "Furkan": 21500123};
// Initialize
   print(myArray["Eylul"]); // Get the value for a given key
   myArray["Etga"] = 21704567; // Add a new element
   print(myArray);

   myArray.remove("Serhat"); // Remove an element
   print(myArray);

   myArray["Eylul"] = 21894459; // Modify the value of an existing
element
   print(myArray);

   print(myArray.containsKey("Eylul")); // Search for the existence of
a key
   print(myArray.containsKey("Serhat")); // Search for the existence of
a key
```

```
   print(myArray.containsValue(21600904)); // Search for the existence
of a value
   print(myArray.containsValue(21900902)); // Search for the existence
of a value

   myArray.forEach((k, v) { // Loop through an associative array, apply
a function, called foo, which simply prints the key-value pair
       foo(k,v);
   });

}
```

1.1.2 Output

```
21602019
{Berdan: 21600904, Omer: 21100999, Eylul: 21602019, Serhat: 21704354, Gunes:
21602271, Furkan: 21500123, Etga: 21704567}
{Berdan: 21600904, Omer: 21100999, Eylul: 21602019, Gunes: 21602271, Furkan:
21500123, Etga: 21704567}
{Berdan: 21600904, Omer: 21100999, Eylul: 21894459, Gunes: 21602271, Furkan:
21500123, Etga: 21704567}
true
false
true
false
Berdan: 21600904
Omer: 21100999
Eylul: 21894459
Gunes: 21602271
Furkan: 21500123
Etga: 21704567
```

1.2 Javascript

1.2.1 Code

```
function foo(k,v)
{
   console.log(k, v);
}
```

```
var myArray = {"Berdan":21600904, "Omer":21100999, "Eylul":21602019,
"Serhat":21704354, "Gunes":21602271, "Furkan":21500123}; // Initialize
console.log(myArray["Eylul"]); // Get the value for a given key

myArray["Etga"] = 21704567; // Add a new element
console.log(myArray);

delete myArray.Serhat; // Remove an element
console.log(myArray);

myArray["Eylul"] = 21894459; // Modify the value of an existing element
console.log(myArray);

console.log("Eylul" in myArray); // Search for the existence of a key
console.log("Serhat" in myArray); // Search for the existence of a key

console.log(Object.values(myArray).includes(21600904)); // Search for
the existence of a value
console.log(Object.values(myArray).includes(21908753)); // Search for
the existence of a value

for(var x in myArray)
   foo(x, myArray[x]);
```

1.2.2 Output

```
21602019
{
  Berdan: 21600904,
  Omer: 21100999,
  Eylul: 21602019,
  Serhat: 21704354,
  Gunes: 21602271,
  Furkan: 21500123,
  Etga: 21704567
}
{
  Berdan: 21600904,
  Omer: 21100999,
  Eylul: 21602019,
  Gunes: 21602271,
  Furkan: 21500123,
  Etga: 21704567
}
{
  Berdan: 21600904,
  Omer: 21100999,
  Eylul: 21894459,
  Gunes: 21602271,
  Furkan: 21500123,
  Etga: 21704567
}
true
false
true
false
Berdan 21600904
Omer 21100999
Eylul 21894459
Gunes 21602271
Furkan 21500123
Etga 21704567
```

1.3 Lua

1.3.1 Code

```lua
function contains(list, x)
   for _, v in pairs(list) do
       if v == x then return true end
   end
   return false
end

function foo(k,v)
   print(k,v)
end

myArray = {["Berdan"] = 21600904, ["Omer"] = 21100999, ["Eylul"] =
21602019, ["Serhat"] = 21704354, ["Gunes"] = 21602271, ["Furkan"] =
21500123} -- Initialize
print(myArray["Eylul"]) -- Get the value for a given key

myArray["Etga"] = 21704567 -- Add a new element
print(myArray["Etga"])
```

```
myArray["Serhat"] = nil -- Remove an element
print(myArray["Serhat"])

myArray["Eylul"] = 21894459 -- Modify the value of an existing element
print(myArray["Eylul"])

print(myArray["Eylul"] ~= nil) -- Search for the existence of a key
print(myArray["Serhat"] ~= nil) -- Search for the existence of a key

print(contains(myArray, 21600904)) -- Search for the existence of a
value
print(contains(myArray, 21901234)) -- Search for the existence of a
value

for k,v in pairs(myArray) do -- Loop through an associative array,
apply a function, called foo, which simply prints the key-value pair
   foo(k,v)
end
```

1.3.2 Output

```
21602019
21704567
nil
21894459
true
false
true
false
Omer   21100999
Furkan  21500123
Etga    21704567
Gunes   21602271
Berdan  21600904
Eylul   21894459
```

1.4 PHP

1.4.1 Code

```
<?php
function foo($k,$v)
```

```
{
   echo $k . ": " . $v . "\n";
}

$myArray = array("Berdan" => 21600904, "Omer" => 21100999, "Eylul" =>
21602019, "Serhat" => 21704354, "Gunes" => 21602271, "Furkan" =>
21500123); // Initialize
echo $myArray["Eylul"]."\n"; // Get the value for a given key

$myArray["Etga"] = 21704567; // Add a new element
print implode(", ",$myArray)."\n";

unset($myArray["Serhat"]); // Remove an element
print implode(", ",$myArray)."\n";

$myArray["Eylul"] = 21894459; // Modify the value of an existing
element
print implode(", ",$myArray)."\n";

if(array_key_exists("Eylul", $myArray)) echo "true\n"; else echo
"false\n"; // Search for the existence of a key
if(array_key_exists("Serhat", $myArray)) echo "true\n"; else echo
"false\n"; // Search for the existence of a key

if(in_array(21600904, $myArray)) echo "true\n"; else echo "false\n"; //
Search for the existence of a value
if(in_array(21904562, $myArray)) echo "true\n"; else echo "false\n"; //
Search for the existence of a value

foreach($myArray as $k=>$v) { // Loop through an associative array,
apply a function, called foo, which simply prints the key-value pair
   foo($k,$v);
}
?>
```

1.4.2 Output

```
21602019
21600904, 21100999, 21602019, 21704354, 21602271, 21500123, 21704567
21600904, 21100999, 21602019, 21602271, 21500123, 21704567
21600904, 21100999, 21894459, 21602271, 21500123, 21704567
true
```

```
false
true
false
Berdan: 21600904
Omer: 21100999
Eylul: 21894459
Gunes: 21602271
Furkan: 21500123
Etga: 21704567
```

1.5 Python

1.5.1 Code

```python
def foo(i, myArray):
   key = list(myArray.keys())[i]
   print(key, myArray[key])


myArray = {"Berdan": 21600904, "Omer": 21100999, "Eylul": 21602019,
"Serhat": 21704354, "Gunes": 21602271, "Furkan": 21500123} # Initialize
print(myArray["Eylul"]) # Get the value for a given key
myArray["Etga"] = 21704567 # Add a new element
print(myArray)
myArray.pop("Serhat") # Remove an element
print(myArray)
myArray["Eylul"] = 21894459 # Modify the value of an existing element
print(myArray)

print("Eylul" in myArray)# Search for the existence of a key
print("Serhat" in myArray)# Search for the existence of a key

print(21600904 in myArray.values()) # Search for the existence of a
value
print(21900123 in myArray.values()) # Search for the existence of a
value



for i in range(0,len(myArray)): # Loop through an associative array,
apply a function, called foo, which simply prints the key-value pair
   foo(i,myArray)
```

1.5.2 Output

```
21602019
{'Berdan': 21600904, 'Omer': 21100999, 'Eylul': 21602019, 'Serhat': 21704354, 'Gunes':
21602271, 'Furkan': 21500123, 'Etga': 21704567}
{'Berdan': 21600904, 'Omer': 21100999, 'Eylul': 21602019, 'Gunes': 21602271, 'Furkan':
21500123, 'Etga': 21704567}
{'Berdan': 21600904, 'Omer': 21100999, 'Eylul': 21894459, 'Gunes': 21602271, 'Furkan':
21500123, 'Etga': 21704567}
True
False
True
False
Berdan 21600904
Omer 21100999
Eylul 21894459
Gunes 21602271
Furkan 21500123
Etga 21704567
```

1.6 Ruby

1.6.1 Code

```ruby
def foo(key, myArray)
   print key, ": ", myArray[key], "\n"
end

myArray = {"Berdan" => 21600904, "Omer" => 21100999, "Eylul" =>
21602019, "Serhat" => 21704354, "Gunes" => 21602271, "Furkan" =>
21500123} # Initialize
puts myArray["Eylul"] # Get the value for a given key
myArray["Etga"] = 21704567 # Add a new element
print myArray, "\n"

myArray.delete("Serhat") # Remove an element
print myArray, "\n"

myArray["Eylul"] = 21894459 # Modify the value of an existing element
print myArray, "\n"
```

```ruby
puts myArray.has_key?("Eylul") # Search for the existence of a key
puts myArray.has_key?("Serhat") # Search for the existence of a key

puts myArray.has_value?(21600904) # Search for the existence of a value
puts myArray.has_value?(21904567) # Search for the existence of a value

for key in myArray.keys # Loop through an associative array, apply a
function, called foo, which simply prints the key-value pair
   foo(key, myArray)
end
```

1.6.2 Output

```
21602019
{"Berdan"=>21600904, "Omer"=>21100999, "Eylul"=>21602019, "Serhat"=>21704354,
"Gunes"=>21602271, "Furkan"=>21500123, "Etga"=>21704567}
{"Berdan"=>21600904, "Omer"=>21100999, "Eylul"=>21602019, "Gunes"=>21602271,
"Furkan"=>21500123, "Etga"=>21704567}
{"Berdan"=>21600904, "Omer"=>21100999, "Eylul"=>21894459, "Gunes"=>21602271,
"Furkan"=>21500123, "Etga"=>21704567}
true
false
true
false
Berdan: 21600904
Omer: 21100999
Eylul: 21894459
Gunes: 21602271
Furkan: 21500123
Etga: 21704567
```

1.7 Rust

1.7.1 Code

```rust
use std::collections::HashMap;
fn contains_value(map: HashMap<&str, i32>, val: i32) -> bool
{
   for i in map.values()
   {
       if i == &val
```

```
        {
            return true;
        }

    }
    return false;
}

fn foo( k: &str, v: i32 )
{
    println!("{}: {}", k, v);
}

fn main()
{
    let mut my_array = HashMap::new(); // Initialize

    my_array.insert("Berdan", 21600904); // Add a new element
    my_array.insert("Omer", 21100999); // Add a new element
    my_array.insert("Eylul", 21602019); // Add a new element
    my_array.insert("Serhat", 21704354); // Add a new element
    my_array.insert("Gunes", 21602271); // Add a new element
    my_array.insert("Furkan", 21500123); // Add a new element

    println!("{}", my_array["Eylul"]); // Get the value for a given key

    my_array.insert("Etga", 21704567); // Add a new element

    my_array.remove("Serhat"); // Remove an element

    //my_array["Eylul"] = 21894459; // Modify the value of an existing
element

    println!("{}", my_array.contains_key("Eylul")); // Search for the
existence of a key
    println!("{}", my_array.contains_key("Serhat")); // Search for the
existence of a key

    println!("{}", contains_value(my_array.clone(), 21600904) ); //
Search for the existence of a value
    println!("{}", contains_value(my_array.clone(), 21903456) ); //
Search for the existence of a value
```

```
   for (key, value) in my_array { // Loop through an associative array,
apply a function, called foo, which simply prints the key-value pair
       foo(key,value);
   }
}
```

1.7.2 Output

```
21602019
true
false
true
false
Gunes: 21602271
Eylul: 21602019
Furkan: 21500123
Omer: 21100999
Etga: 21704567
Berdan: 21600904
```

2. Language Evaluations

The evaluations for the languages are as below. In my opinion, the most useful language to work with associative arrays is Python since it is easy to read and write. Dictionary data type is easy to use and understand. Also there is a variety of functions that can be used for any operations and there are lots of documentation online. So a Python user can easily read or write a program that use associative arrays using Python.

2.1 Dart

2.1.1 Readability

- Usage of an associative array is simple in Dart since there are only a small amount of functions required to do all the associative array operations.
- Usage of an associative array is orthogonal since every possible combination for the keys and values are valid.
- Dart language has an easy to use syntax for associative arrays. All keywords to functions are meaningful and similar to the well-known functions.

2.1.2 Writability

- As discussed above, usage of an associative array is simple and orthogonal in Dart. This makes Dart writable too.
- Associative arrays in Dart supports abstraction since a key or a value of a dictionary can be a class.
- Associative arrays in Dart are expressive since operations like adding and removing can be done with predefined functions.

2.2 Javascript

2.2.1 Readability

- Usage of an associative array is simple in Javascript since there are not many parentheses needed and there are only a small amount of functions required to do all the associative array operations.
- Usage of an associative array is orthogonal since every possible combination for the keys and values are valid.
- Python language has its own dictionary data type as associative arrays.
- Python language has an easy to use syntax for associative arrays. All keywords to functions are meaningful and similar to the well-known functions.

2.2.2 Writability

- Associative arrays in Javascript are expressive since operations like adding and removing can be done with predefined functions.

2.3 Lua

2.3.1 Readability

- Usage of an associative array is simple in Lua since there are not many parentheses needed and there are only a small amount of functions required to do all the associative array operations.
- Usage of an associative array is orthogonal since every possible combination for the keys and values are valid.
- Lua does not have a data type for associative arrays. This reduces readability.
- Lua language has an easy to use syntax for associative arrays. All keywords to functions are meaningful and similar to the well-known functions.

2.3.2 Writability

- Associative arrays in Javascript are expressive since operations like adding and removing can be done with predefined functions.

2.4 PHP

2.4.1 Readability

- Associative arrays are not simple in PHP.
- Javascript does not have a data type for associative arrays. This reduces readability.
- PHP language has a hard to use syntax for associative arrays. Some functions are hard to understand since they are not self-explaining.

2.4.2 Writability

- Associative arrays in Dart are expressive since operations like adding and removing can be done with predefined functions.

2.5 Python

2.5.1 Readability

- Usage of an associative array is simple in Python since there are not many parentheses needed and there are only a small amount of functions required to do all the associative array operations.
- Usage of an associative array is orthogonal since every possible combination for the keys and values are valid.
- Python language has its own dictionary data type as associative arrays.
- Python language has an easy to use syntax for associative arrays. All keywords to functions are meaningful and similar to the well-known functions.

2.5.2 Writability

- As discussed above, usage of an associative array is simple and orthogonal in Python
- Associative arrays in Python supports abstraction since a key or a value of a dictionary can be a class.
- Associative arrays in Python are expressive since operations like adding and removing can be done with predefined functions.

2.6 Ruby

2.6.1 Readability

- Usage of an associative array is simple in Ruby since there are not many parentheses needed and there are only a small amount of functions required to do all the associative array operations.
- Usage of an associative array is orthogonal since every possible combination for the keys and values are valid. This increases readability.
- The language has an easy to understand syntax

2.6.2 Writability

- Associative arrays in Ruby supports abstraction since a key or a value of a dictionary can be a class.
- Associative arrays in Ruby are expressive since operations like adding and removing can be done with predefined functions.

2.7 Rust

2.7.1 Readability

- Rust language is not a simple language. Because use of functions and syntax are not similar to well known languages.
- Rust associative arrays are orthogonal since they can get any type of value as keys or values. But every keys and every values has to be same type
- Rust syntax is hard to understand and this reduces readability

2.7.2 Writability

- Hardness of syntax and low simplicity reduces writability too.
- Associative arrays in Rust are expressive since operations like adding and removing can be done with predefined functions.

3. Learning Strategy

I used linux compilers and interpretters for this homework. No online compiler/interpretter is used. The list of the compilers used are as follows:

- Dart SDK 2.10.4

- Node v12.19.0
- Lua 5.4.0
- PHP 7.4.5
- Python 3.8.4
- Ruby 2.7.1p83
- Rustc 1.48.0

While doing this assignment, I started with Python which is the only familiar language to me on the list. I already knew how to use dictionaries in Python and easily implemented the Python source code. Later I installed compilers/interpretters for each language and after that, I researched about associative array usage for each language one by one. I mostly used documentation pages for each language to understand how associative arrays work. For most of the languages, usage of the associative arrays were similar to Python. This made me understand other languages easier too. While learning and understanding, I kept compiling and running codes for each language. This way, I could see my mistakes and correct them easily.