CS315 HOMEWORK 1 REPORT

Onurcan Ataç

22002194

Section 2

Associative Array Implementations

1.Dart

a.Code

```
//dartpad.dev/?

void main()
{
    //initialization

    var tennisPlayer = {
    "name":"Roger",
    "surname":"Federer",
    "age": 41,
    "racket":"Wilson Pro Staff",
    "favorite_shot": "Forehand",
    "backhand_type": "One-handed"
    };

    //get and print the value for key "racket"

    var racket = tennisPlayer["racket"];
    print(racket);

    //add height item to the object

    tennisPlayer["height"] = 185;

    //remove backhand type from the object

    tennisPlayer.remove("backhand_type");

    //change the favorite shot from Forehand to Backhand

    tennisPlayer["favorite_shot"] = "Backhand";

    //check for the existence of age (a key) in the object

    var ageExists = false;

    if(tennisPlayer.containsKey("age"))
    {
        ageExists = true;
    }

    print(ageExists);

    //check for the existence of 185 (a value) in the object

    var valueExists = false;

    if(tennisPlayer.containsValue(185))
    {
        valueExists = true;
    }

    print(valueExists);

    //apply a function foo that prints the key-value pair

    foo(var key, var value)
    {
        print("$key" + " : " + "$value");
    }

    tennisPlayer.forEach((key, value)
    {
        foo(key, value);
    });
}
```

Code is explained by comments. Changes, additions and deletions can be checked from the output of the last part, foo.

Code is ran by using https://dartpad.dev/?

b.Output

```
Wilson Pro Staff
true
true
name : Roger
surname : Federer
age : 41
racket : Wilson Pro Staff
favorite_shot : Backhand
height : 185
```

2.Javascript

a.Code

```
<!DOCTYPE html>
<html><head><script>//onecompiler.com/javascript

    //initialization

    var tennisPlayer = {
      name: "Roger",
      surname: "Federer",
      age: 41,
      racket: "Wilson Pro Staff",
      favorite_shot: "Forehand",
      backhand_type: "One-handed",
    };

    //get and print the value for key "racket"

    var racket = tennisPlayer.racket;
    console.log(racket);

    //add height item to the object

    tennisPlayer.height = 185;

    //remove backhand type from the object

    delete tennisPlayer.backhand_type;

    //change the favorite shot from Forehand to Backhand

    tennisPlayer.favorite_shot = "Backhand";

    //check for the existence of age (a key) in the object

    var propExists = false;

    if (tennisPlayer.hasOwnProperty("age")) {
      propExists = true;
    }

    console.log(propExists);

    //check for the existence of 185 (a value) in the object

    var valExists = false;
```

```
if (Object.values(tennisPlayer).includes(185)) {
  valExists = true;
}

console.log(valExists);

//apply a function foo that prints the key-value pair

function foo(key, value) {
  console.log(key + ": " + value);
}

for (var [key, value] of Object.entries(tennisPlayer)) {
  foo(key, value);
}
</script></head><body>output is printed on the console</body></html>
```

Code is explained by comments. Changes, additions and deletions can be checked from the output of the last part, foo. Javascript code is submitted in an HTML file as wanted.

Code is ran by using https://onecompiler.com/javascript, in the ".js" extension. In HTML version, output comes up on the console when ran.

b.Output

```
Wilson Pro Staff
true
true
name: Roger
surname: Federer
age: 41
racket: Wilson Pro Staff
favorite_shot: Backhand
height: 185
```

3.Lua

a.Code

```lua
--onecompiler.com/lua

--initialization

tennisPlayer = {
  name="Roger",
  surname="Federer",
  age = 41,
  racket = "Wilson Pro Staff",
  favorite_shot = "Forehand",
  backhand_type = "One-handed"}

--get and print the value for key "racket"

racket = tennisPlayer.racket
print(racket)

--add height item to the table

tennisPlayer.height = 185

--remove backhand type from the table
--setting an item to nil is the only convention I found in order to remove a
key from the table

tennisPlayer.backhand_type = nil

--change the favorite shot from Forehand to Backhand

tennisPlayer.favorite_shot = "Backhand"

--check for the existence of age (a key) in the table

ageExists = false

if tennisPlayer.age ~= nil
then ageExists = true
end

print(ageExists)

--check for the existence of 185 (a value) in the table

numberExists = false
```

```lua
for _, value in pairs(tennisPlayer) do
if value == 185 then
  numberExists = true
end
end

print(numberExists)

--apply a function foo that prints the key-value pair

function foo(key, value)
print(key,value)
end

for key, value in pairs(tennisPlayer)
do foo(key, value) end
```

Code is explained by comments. Changes, additions and deletions can be checked from the output of the last part, foo.

Code is ran by using https://onecompiler.com/lua

b.Output

```
Wilson Pro Staff
true
true
age     41
favorite_shot  Backhand
name    Roger
surname Federer
height  185
racket  Wilson Pro Staff
```

4.PHP

a.Code

```php
<?php
//onecompiler.com/php

//initialization

$tennisPlayer = array(
    "name" => "Roger",
    "surname" => "Federer",
    "age" => 41,
    "racket" => "Wilson Pro Staff",
    "favorite_shot" => "Forehand",
    "backhand_type" => "One-handed");

//get and print the value for key "racket"

$racket = $tennisPlayer["racket"];
echo $racket;
echo "\r\n";

//add height item to the associative array

$tennisPlayer["height"] = 185;

//remove backhand type from the associative array

unset($tennisPlayer["backhand_type"]);

//change the favorite shot from Forehand to Backhand

$tennisPlayer["favorite_shot"] = "Backhand";

//check for the existence of age (a key) in the associative array

$ageExists = false;

if(array_key_exists("age", $tennisPlayer))
{
    $ageExists = true;
}

if ($ageExists == true)
{
  echo "true";
}
```

```php
else
{
  echo "false";
}

echo "\r\n";

//check for the existence of 185 (a value) in the associative array

$numberExists = in_array( 185, $tennisPlayer);

if ($numberExists == true)
{
  echo "true";
}
else
{
  echo "false";
}

echo "\r\n";

//apply a function foo that prints the key-value pair

function foo($key, $value)
{
  echo $key . ", ". $value . " \n";
}

foreach($tennisPlayer as $key => $value)
{
  foo($key, $value);
}
?>
```

Code is explained by comments. Changes, additions and deletions can be checked from the output of the last part, foo.

Code is ran by using https://onecompiler.com/php

b.Output

```
Wilson Pro Staff
true
true
name, Roger
surname, Federer
```

```
age, 41
racket, Wilson Pro Staff
favorite_shot, Backhand
height, 185
```

5.Python

a.Code

```python
#onecompiler.com/python

#initialization

tennisPlayer = {
    "name":"Roger",
    "surname":"Federer",
    "age": 41,
    "racket":"Wilson Pro Staff",
    "favorite shot": "Forehand",
    "backhand type": "One-handed"
}

#get and print the value for key "racket"

racketOfPlayer = tennisPlayer.get("racket")
print(racketOfPlayer)

#add height item to the dictionary

tennisPlayer["height"] = 185

#remove backhand type from the dictionary

tennisPlayer.pop("backhand type")

#change the favorite shot from Forehand to Backhand

tennisPlayer["favorite shot"] = "Backhand"

#check for the existence of age (a key) in the dictionary

agePresent = False

for key in tennisPlayer:
    if key == "age":
        agePresent = True
```

```
print(agePresent)

#check for the existence of 185 (a value) in the dictionary

numberPresent = False

for value in tennisPlayer.values():
    if value == 185:
        numberPresent = True

print(numberPresent)

#apply a function foo that prints the key-value pair

def foo(key, value):
    print(key, value)

for key, value in tennisPlayer.items():
    foo(key, value)
```

Code is explained by comments. Changes, additions and deletions can be checked from the output of the last part, foo.

Code is ran by using https://onecompiler.com/python

b.Output

```
Wilson Pro Staff
True
True
name Roger
surname Federer
age 41
racket Wilson Pro Staff
favorite shot Backhand
height 185
```

6.Ruby

a.Code

```ruby
#onecompiler.com/ruby

#initialization

tennisPlayer = {
    "name" => "Roger",
    "surname" => "Federer",
    "age" => 41,
    "racket" => "Wilson Pro Staff",
    "favorite_shot" => "Forehand",
    "backhand_type" => "One-handed"
}

#get and print the value for key "racket"

racket = tennisPlayer["racket"]
puts racket

#add height item to the hash

tennisPlayer["height"] = 185

#remove backhand type from the hash

tennisPlayer.delete("backhand_type")

#change the favorite shot from Forehand to Backhand

tennisPlayer["favorite_shot"] = "Backhand"

#check for the existence of age (a key) in the hash

ageExists=false

if tennisPlayer.has_key?("age")
  ageExists = true
end

puts ageExists

#check for the existence of 185 (a value) in the hash

numberExists=false
```

```
if tennisPlayer.has_value?(185)
  numberExists = true
end

puts numberExists

#apply a function foo that prints the key-value pair

  def foo(key, value)
    puts "#{key} , #{value}"
  end

  tennisPlayer.each do |key,value|
    foo(key,value)
  end
```

Code is explained by comments. Changes, additions and deletions can be checked from the output of the last part, foo.

Code is ran by using https://onecompiler.com/ruby

“”

b.Output

```
Wilson Pro Staff
true
true
name , Roger
surname , Federer
age , 41
racket , Wilson Pro Staff
favorite_shot , Backhand
height , 185
```

7.Rust

a.Code

```
//onecompiler.com/rust

use std::collections::HashMap;

fn main()
{
    //initialization

    let mut tennis_player = HashMap::new();

    //mismatched types error if I try int as a value

    tennis_player.insert(
      "name",
      "Roger"
    );
      tennis_player.insert(
      "surname",
      "Federer"
    );
      tennis_player.insert(
      "age",
      "41"
    );
      tennis_player.insert(
      "racket",
      "Wilson Pro Staff"
    );
      tennis_player.insert(
      "favorite_shot",
      "Forehand"
    );
      tennis_player.insert(
      "backhand_type",
      "One-handed"
    );

    //get and print the value for key "racket"

    match tennis_player.get("racket")
    {
        Some(racket) => println!("{}",racket),
        None => println!("Racket not found.")
    }
    //add height item to the object

    //mismatched types error if I try int as a value

    tennis_player.insert(
      "height",
      "185"
    );

    //remove backhand type from the object

    tennis_player.remove("backhand_type");

    //change the favorite shot from Forehand to Backhand

    tennis_player.insert("favorite_shot", "Backhand");

    //check for the existence of age (a key) in the object

    let age_exists = tennis_player.contains_key("age");
    println!("{}", age_exists);

    //check for the existence of 185 (a value) in the object

    let height_exists = tennis_player.values().any(|&value| value == "185");
    println!("{}", height_exists);

    //apply a function foo that prints the key-value pair

    fn foo(key: &str, value: &str)
    {
      println!("{} : {}",key, value);
    }

    for (key, value) in &tennis_player {
      foo(key, value);
    }
}
```

Code is explained by comments. Changes, additions and deletions can be checked from the output of the last part, foo.

Code is ran by using https://onecompiler.com/rust

b.Output

```
Wilson Pro Staff
true
true
surname : Federer
age : 41
racket : Wilson Pro Staff
favorite_shot : Backhand
height : 185
name : Roger
```

Evaluation of Languages for Associative Array Data Structures

For me, important factors were in line with readability and writability. Popularity of languages were also important since more popularity makes it easy to quickly access more features from the internet. In my opinion, the worst language to use for associative array operations out of languages we used in the homework is Rust. I couldn't find a way of creating an associative array with keys and values already in it, so I had to insert keys and values one by one. Moreover, while inserting, I couldn't insert a string key with integer value, I got the "mismatched types error". Also, it didn't let me use camel case notation on variables like I'm used to, it forced me to write with snake case notation, which decreases readability and writability. For PHP, Lua and Javascript, there were functions present in order to find specific keys and values, so they were preferable and average. It was really easy to write with Ruby, Python and Dart. They contained laborsaving methods, their syntax of basic functions were familiar for me and it did make reading and writing very easy. Especially on Dart, I really enjoyed programming. Hence, I would probably select Dart or Python in order to use on associative array operations.

Learning Strategies

For his task, I decided to start with Python as I already know some Python, and determine my learning strategy after completing the program with the first language. After I completed Python, I realized that there might be enough explanation on the internet about associative arrays in other languages as well. As I already know what are the conditions to be an associative array (as given in the lectures and start of the homework document) I thought that I can separate what associative arrays in different languages are considering what comes up on the internet. I completed my homework repeating this routine, searching with different key words if I couldn't find what I wanted. Even though it was hard to find information in some languages such as Rust and Lua, every information I needed was present on the internet. At some parts, I also experimented on the online compilers to check if some syntax that might work, using my experience from other programming languages.

The online compilers that I used are given in the report after code parts of every language.