CS315

HOMEWORK 2

HACI ÇAKIN

21802641

SECTION 2

1) C

In C, iterations are tested one by one. There are while, for and do-while statements. All loops continues until condition is failed or break is called. In for and nested loops, variable controls the statement by incrementing this variable. In second part, array is tested for iteration. The last part, in for loop, element is gotten by using variable directly. What I mean is that if variable is equal to 2, then it directly goes that address. Same think happens in while because it also uses variables to iterate.

1.A) Iteration statements provided

In C provided iterative statements are while, for and do-while.

Ex.

```
while( trace < 5 ){
   printf("Value of trace : %d in while loop\n", trace);
   trace++;
}
```

Output.

```
Value of trace : 0 in while loop
Value of trace : 1 in while loop
Value of trace : 2 in while loop
Value of trace : 3 in while loop
Value of trace : 4 in while loop
```

```
for(int i = 0; i < 5; i++){
   printf("Value of i : %d in for loop\n", i);
}
```

Output.

```
Value of i : 0 in for loop
Value of i : 1 in for loop
```

Value of i : 2 in for loop

Value of i : 3 in for loop

Value of i : 4 in for loop

```
do{
   printf("Value of number : %d in do-while loop\n", number);
   number++;
}while(number < 5);
```

Output.

Value of number : 0 in do-while loop

Value of number : 1 in do-while loop

Value of number : 2 in do-while loop

Value of number : 3 in do-while loop

Value of number : 4 in do-while loop

1.B) Data structures suitable for iteration

In C, built in data structures that are suitable for iteration are arrays.

Ex.

```
char array1[] = {'h','a','c','i'};
int array2[] = {2,3,5,1,6,3};
for(int i = 0; i < 4; i++){
   printf("%d. letter of haci is %c \n", i+1,array1[i]);
}
```

Output.

1. letter of haci is h

2. letter of haci is a

3. letter of haci is c

4. letter of haci is I

```
trace = 0;
while(array2[trace] != 6){
    printf("6 is not found until %d. element \n",trace);
    trace++;
}
```

Output.

```
6 is not found until 0. element
6 is not found until 1. element
6 is not found until 2. element
6 is not found until 3. element
```

1.C) The way the next item is accessed

Ex.

```
int num1 = 0;
while ( num1 < 11 ){
    if( num1 == 6 ){
        printf("Reached  '6'\n");
        num1 += 2;
    }
    num1 = num1 + 1;
    printf(" Turn %d times", num1);
    printf("\n");
}
```

Output.

```
Turn 2 times
Turn 3 times
Turn 4 times
Turn 5 times
Turn 6 times
Reached  '6'
Turn 9 times
Turn 10 times
Turn 11 times
```

```
for(int a = 0; a < 10; a = a+2){
    printf("Current value is %d \n",a );
}
```

Output.

Current value is 0

Current value is 2

Current value is 4

Current value is 6

Current value is 8

2) JAVASCRIPT

In javascript, iterations are tested one by one. In this language, there are while, for, do-while, for/in and for/of for iterations. In second part, I see that sets, arrays and objects can be used for iterations. In the last part, there are two ways in javascript to reach element. First way is, reaching element by directly its address by using variable. Second way is that like in forEach loop travel the all elements one by one.

2.A) Iteration statements provided

In js, there are while, for, do-while, for/in, for/of, forEach to iterate.

Ex.

```
number1 = 0;
var bool1 = true;
while (bool1) {
  if (number1 == 4) {
    bool1 = false;
    console.log("number1 reached to 4");
    break;
  }
  console.log("number1 = " + number1);
  number1++;
```

```
    }
```

Output.

```
number1 = 0
number1 = 1
number1 = 2
number1 = 3
number1 reached to 4
```

```
    for (i = 0; i < 5; i++) {
      if (i == 4) {
        console.log("i reached to 4");
      } else {
        console.log("i = " + i);
      }
    }
```

Output.

```
i = 0
i = 1
i = 2
i = 3
i reached to 4
```

```
    bool1 = true;
    number2 = 0;
    do {
      if (number2 == 4) {
        console.log("number2 reached to 4");
        bool1 = false;
        break;
      }
      console.log("number2 = " + number2);
      number2++;
    } while (bool1);
```

Output.

number2 = 0

number2 = 1

number2 = 2

number2 = 3

number2 reached to 4

```
numArray = [0, 1, 2, 3, 4];
for (temp of numArray) {
  console.log("temp = " + temp);
}
```

Output.

temp = 0

temp = 1

temp = 2

temp = 3

temp = 4

```
var dates = { birhday: 24, bilkent: 1984, FB: 1907 };
for (temp in dates) {
  console.log("temp = ", temp);
}
```

Output.

temp = birhday

temp = bilkent

temp = FB

```
var languages = ["tr", "en", "fr"];
languages.forEach((item, index, arr) => console.log(item));
```

Output.

tr

en

fr

2.B) Data structures suitable for iteration

In javascript, arrays and objects can be used as a data structure which can be iterable.

Ex.

```
numArray2 = [0, 1, 2, 3, 4];
for (var temp of numArray2) {
    console.log("temp = " + temp);
 }
```

Output.

```
temp = 0
temp = 1
temp = 2
temp = 3
temp = 4
```

```
 numSet = new Set();
 for (i = 0; i < 5; i++) {
    numSet.add(i);
  }
 for (i = 0; i < 5; i++) {
    console.log(numSet.has(i));
  }
```

Output.

```
True
True
True
True
True
```

```
  myObject = { name: "haci", age: 21, dept: "cs" };
  for (temp in myObject) {
    console.log("temp = " + temp);
   }
```

Output.

temp = name

temp = age

temp = dept

2.C) The way the next item is accessed

Ex.

```
    while (number3 < 10) {
      console.log("number3 = " + number3);
      number3 = number3 + 2;
    }
```

Output.

number3 = 0

number3 = 2

number3 = 4

number3 = 6

number3 = 8

```
    for (i = 0; i < 10; i++) {
      console.log("i = " + i);
      i++;
    }
```

Output.

i = 0

i = 2

i = 4

i = 6

i = 8

3) PHP

In php, iterations are tested one by one. In this language, there are while, for, do-while, for iterations. In second part, I see that arrays can be used for iterations. In the last part, there is one way in php to reach element. The way is, reaching element by directly its address by using variable.

3.A) Iteration statements provided

In php, there are while, for, forEach and do-while for iteration

Ex.

```
while($bool1){
    if($num1 == 5){
        printf("num1 reached to 5\n");
        break;
    }
    printf("num1 = %d \n", $num1);
    $num1++;
}
```

Output.

num1 = 0
num1 = 1
num1 = 2
num1 = 3
num1 = 4
num1 reached to 5

```
$num2 = 0;
do{
    $num2+=2;
    printf("num2 = %d \n", $num2);
} while( $num2 < 10 );
```

Output.

num2 = 2
num2 = 4
num2 = 6
num2 = 8
num2 = 10

```
for($k = 0; $k < 6; $k++){
    if($k ==4){
        printf("Reached to 4\n");
```

```
        break;
    }
    printf("k = %d \n", $k);
}
```

Output.

```
k = 0
k = 1
k = 2
k = 3
Reached to 4
```

```
$arrayNum = array(0, 1, 2);
foreach ($arrayNum as $value) {
   $value = $value + 4;
   printf("value = %d \n", $value);
   printf( "\n");
}
```

Output.

```
value = 4
value = 5
value = 6
```

3.B) Data structures suitable for iteration

In php, arrays can be used in iterative statements.

Ex.

```
$arrayNum = array(0, 1, 2, 3, 4, 5);
foreach ($arrayNum as $value) {
    printf("value = %d \n", $value);
 }
```

Output.

```
value = 0
value = 1
value = 2
value = 3
```

value = 4

value = 5

```
$birthDay = array ("day"  => 24, "month" => 02, "year"   => 2000 );
 foreach ($birthDay as $name => $value) {
    printf("value = %d \n", $value);
 }
```

Output.

value = 24

value = 2

value = 2000

3.C) The way the next item is accessed

Ex.

```
$num3 = 0;
     while( $num3 < 10 ) {
     if ( $num3 == 4){
            printf("num3 reached 4\n");
           break;
         }
     printf("num3 = %d \n", $num3);
     $num3 = $num3 + 2;


     }
```

Output.

num3 = 0

num3 = 2

num3 reached 4

```
     for( $m= 0; $m< 10 ; $m++){
       if( $m == 4 ){
            printf("m reached 4\n");
```

```
            break;
        }
        printf("m = %d \n", $m);
        $m = $m + 1;
    }
```

Output.

num3 = 0

num3 = 2

num3 reached 4

4) RUST

In rust, possibilities are tested one by one. In this language, there are while, for, loop statements for iterations. In second part, I see that arrays and vectors can be used for iterations. In the last part, there is one way in rust to reach element. The way is, reaching element by directly its address by using variable.

4.A) Iteration statements provided

In rust, there are while, for and loop statements.

Ex.

```
let mut num = 0;
while num < 5 {
    if num != 4{
        println!("{} is the current value in while", num);
    }else{
        println!("while reached to 4\n");
        break;
    }
    num = num + 1;
}
```

Output.

0 is the current value in while

1 is the current value in while

2 is the current value in while

3 is the current value in while

while reached to 4

```
let mut num1 = 0;
loop{
    if num1 == 4 {
        println!("loop reached to 4\n");
        break;
    }
    println!("{} is the current value in loop",num1);
    num1 = num1 + 1;
}
```

Output.

0 is the current value in loop

1 is the current value in loop

2 is the current value in loop

3 is the current value in loop

loop reached to 4

```
for num2 in 0..4{
    println!("{} is the current value in for",num2);
}
```

Output.

0 is the current value in for

1 is the current value in for

2 is the current value in for

3 is the current value in for

for reached to 4

4.B) Data structures suitable for iteration

In Rust, there is just array for this operation.

Ex.

```
let array2 = [1907,24,02,2000];
for num4 in array2.iter(){
    println!("Element is: {}", num4 );
}
```

Output.

```
Element is: 1907
Element is: 24
Element is: 2
Element is: 2000
```

4.C) The way the next item is accessed

Ex.

```
let mut num5 = 0;
  while num5 < 11 {
     if num5 == 6 {
       println!("num5 Reached to 6");
       break
   }
   println!("num5 =  {}", num5);
   num5 = num5 + 2;
 }
```

Output.

```
num5 =  0
num5 =  2
num5 =  4
num5 Reached  to 6
```

5) GO

In rust, possibilities are tested one by one. In this language, there are while, for, forEach statements for iterations. In second part, I see that arrays

and slices can be used for iterations. In the last part, there are two ways in go to reach element. The way is, reaching element by directly its address by using variable. Second one is the traveling all the elements one by one by using forEach

5.A) Iteration statements provided

In Goland, there are while, for and forEach.

Ex.

```
var bool1 = true
trace := 0
for bool1 {
    if trace == 5 {
        fmt.Println("Trace reached to 5 \n")
        break
    }
    fmt.Println("Current value in while loop is ", trace)
    trace = trace + 1
}
```

Output.

```
Current value in while loop is  0
Current value in while loop is  1
Current value in while loop is  2
Current value in while loop is  3
Current value in while loop is  4
Trace reached to 5
```

```
for i :=0; i < 6; i++ {
    fmt.Println("Current value in for loop is ", i)
}
```

Output.

```
Current value in for loop is  0
Current value in for loop is  1
Current value in for loop is  2
Current value in for loop is  3
```

```
Current value in for loop is  4
Current value in for loop is  5
```

```
arrayEx := []int64{ 1907,2000,1883}
for place,value := range arrayEx {
    fmt.Println("Place : ", place , " Value : ", value)
}
```

```
Output.
Place :  0  Value :  1907
Place :  1  Value :  2000
Place :  2  Value :  1883
```

5.B) Data structures suitable for iteration

In Goland, there are arrays and slices that can be used for iteration.

```
Ex.
```

```
array1 := [4]int64{24,02,2000,1907}
fmt.Println("Array is ", array1)
for i := 0; i < 4; i++ {
    fmt.Println(i+1,". element is : ", array1[i])
}
```

```
Output.
Array is  [24 2 2000 1907]
1 . element is :  24
2 . element is :  2
3 . element is :  2000
4 . element is :  1907
```

```
slice1 := [4]string{"KING","OF","THE","NOTRH"}
for place,value := range slice1 {
    fmt.Println("Place : ", place , " Value : ", value)
}
```

```
Output.
Place :  0  Value :  KING
```

Place : 1 Value : OF

Place : 2 Value : THE

Place : 3 Value : NOTRH

5.C) The way the next item is accessed

Ex.

```
num2:= 0
   for num2 < 8 {
      if  num2 == 6 {
         fmt.Println("Reached  '6'")
         num2 = num2 + 2
         break
      }
      num2= num2 + 1
      fmt.Println(" Turn ", num2 , " times")
   }
```

Output.

Turn 1 times

Turn 2 times

Turn 3 times

Turn 4 times

Turn 5 times

Turn 6 times

Reached '6'

```
   for k := 0; k < 11; k = k +2 {
      if  k == 6 {
         fmt.Println("Reached  '6'")
         break
      }
      fmt.Println(" Turn ", k , " times")
   }
```

Output.

Turn 1 times

Turn 3 times

Turn 5 times

Turn 7 times

Turn 9 times

6) PYTHON

In python, possibilities are tested one by one. In this language, there are while, for and nested loop statements for iterations. In second part, I see that set, tuple, dictionary, and List can be used for iterations. In the last part, there is one way in python to reach element. The way is, reaching element by directly its address by using variable.

6.A) Iteration statements provided

In python, there are while, for and nested loop

Ex.

```
while(check):
    if(number1 == 3):
        print("Reached to 3 \n")
        check = False
        break
    print(number1)
    number1 += 1
```

Output.

0

1

2

Reached to 3

```
for i in range (1,10):
    if(i == 5):
        print("Reached to 5 \n")
        break
```

```
    print(i)
```

Output.

```
1
2
3
4
Reached to 5
```

```
for i in range (1,5):
    for j in range(2,3):
        print(i , " * ", j ," = ", i*j )
```

Output.

```
1  *  2  =  2
2  *  2  =  4
3  *  2  =  6
4  *  2  =  8
```

6.B) Data structures suitable for iteration

In python, there are set, tuple, dictionary, enumerates and lists.

Ex.

```
tupple = ("23","haci",("halil","altay"))
for i in tupple:
    print(i)
```

Output.

```
23
haci
('halil', 'altay')
```

```
list1 = ["23","haci",["halil","altay"]]
for i in range (0,3):
    print (list1[i])
```

Output.

```
23
haci
```

['halil', 'altay']

```
dictionary = {"age":23,"name":"haci","dep":"cs"}
for i in dictionary:
    print(dictionary[i])
```

Output.

23

haci

cs

```
set1 = {"23","haci","cs"}
for i in set1:
    print(i)
```

Output.

cs

23

haci

```
for key, value in enumerate(['cs', '315', 'bilkent', 'Atalar']):
    print(key, value)
```

Output.

0 cs

1 315

2 bilkent

3 Atalar

6.C) The way the next item is accessed

Ex.

```
while( number2 < 10 ):
    print("number2 = " , number2)
    number2 = number2 + 2
```

Output.

number2 = 0
number2 = 2
number2 = 4
number2 = 6
number2 = 8

```
for number3 in range (0,10):
    print("number3 = ", number3)
    number3 = number3 + 3
```

Output.

number3 = 0
number3 = 1
number3 = 2
number3 = 3
number3 = 4
number3 = 5
number3 = 6
number3 = 7
number3 = 8
number3 = 9

7) Compilers

- Php: https://paiza.io/projects/vyadNArc1edxqXKYyHtUmA
- C: https://www.programiz.com/c-programming/online-compiler/
- Python: On Visual Studio Code
- Javascript: On browser's console
- Goland: https://play.golang.org/
- Rust: https://play.rust-lang.org/

8) Learning Methodology

I do not use any source for C, python and javascript because I overall know them. For Go, I looked at the golang basic documentation(https://tour.golang.org/flowcontrol/1). For more details, I looked at geekforgeeks(https://www.geeksforgeeks.org/loop-control-statements-in-go-language/) and stackoverflow. For rust language, I used the official documentation of Rust(https://doc.rust-lang.org/beta/rust-by-example/index.html) (https://doc.rust-lang.org/beta/book/index.html). For the php language, I again used w3school(https://www.w3schools.com/php/php_looping.asp) and official php Manuel(https://www.php.net/manual/en/).

Conclusion

Lastly, in terms of writability, javascript is the best language with respect to loop statements. In terms of built in data structures, python is the best language for writability. As writability increases, these languages loses readability. In terms of writability, php is the worst one is because it either has less number of iterative statements and data structures that is suitable for iteration. But this increase readability and reliability. Using forEach, for/in, for, of like iterations increase the readability and it give change to not declare one more variable for iteration.