İlke Kaş

21803184

Section 1

REPORT

C

1. Iteration statements provided

- **Explanation:** There are three types of iteration statements in C language. These are:
    - for loop
    - while loop
    - do... while loop

For loops are **counter-controlled** loops. Counter-controlled loop means that the loop is definite repetition since the number of iterations is known before the loops begin executing. However, while and do-while loops are not counter-controlled. They are **logically-controlled** loops. Repetition is based on boolean expressions rather than counter. Logically-controlled loops are often called indefinite repetition since the number of iterations is not known before the loops begin executing. Besides these, we can group the loops according to the place where the loop condition is tested, before or after entering the loop body. Since for loop and while loop conditions are tested before the body execution, we call them **pretest.** However, do-while loop is **posttest** loops and that means, the body of the do-while loop will be executed at least once, then the condition will be checked.

- **For loop:**

  Its syntax like in the following:

  for(expression1; expression2 ; expression3)

  loop body

  Loop body can be a single, compound or a null statement. Expression1 is executed once and it generally holds initialization of the counter in C. Besides, it can also declare the counter in C99. Expression2 is executed

before each iteration and it generally holds for the conditions. In C, values that correspond to 0, makes condition false then terminates the loop execution. However, in C99 boolean variables can be used. Values that are different from 0 are considered as true and enable loop execution. Expression 3 is executed at the end of each iteration, it holds the update statement for counters generally such as incrementation or decrementation. All of these expressions are optional in the syntax of the for loop. If the statement is like following:

for(expression1; ; expression3)

Since the conditional expression is absent, C assumes that it is always true and this causes infinite loop. Besides these, multiple statements for expression1 and expression3 can be written by separating statements with comma(,). Sometimes, loop statements do not have a loop body to execute, all of the works are done between the parentheses of the for statement.

- **While loop:**

  Its syntax like in the following:

  while (control_expression)

  loop body

  The conditional expression in the while statement has to be given unlike for loop. Otherwise, it gives errors. While statements can simulate for loop logic. While statements are **pretest logical-controlled loops.** So, we cannot always know the iteration number of the loop. It executes the body as long as the conditional expression is true.

- **DO-While loop:**

  Its syntax like in the following:,

  do

  loop body

  while (control_expression);

Do while statements are **posttest logical-controlled loops.** In do-while, the loop body is executed until the control expression evaluates to false. This means at least once the body of the loop will be executed.

- **Code Segment for FOR LOOP:**

```
int arr[] ={0,1,2,3,4};
int i;
int j;
printf("0 value is considered as false in the second expression\n");
for(i = 4; arr[i]; i--)
{
    printf("arr[%d] : %d\n",i,arr[i]);
}
printf("\n");
i = 0;
printf("First and the third expressions are absent:\n");
int arr3[5] = {0,1,2,3,4};
for( ; arr3[i] <2 ; )
{
      printf("arr3[%d] : %d\n",i,arr3[i]);
      i++;
}
printf("Multiple statements for first and third expressions: \n");
int arr4[5] = {0,1,2,3,4};
for(i= 3, j = 2; i < 5 && j <5; arr4[i] +=1, arr4[j] *=2, i++,j++)
{
    printf("arr4[%d] : %d, arr4[%d]: %d",i,j,arr4[i],arr4[j]);
}
printf("For loop without body: \n");
int arr5[5] = {0,1,2,3,4};
for(i = 1; arr5[i] < 3; arr5[i] *= 2, printf("arr5[%d]: %d\n",i,arr5[i]));
```

- **Result of Execution:**

```
0 value is considered as false in the second expression:
arr[4] :4
arr[3] :3
arr[2] :2
arr[1] :1

First and teh third expressions are absent:
arr3[0]:0
arr3[1]:1
Multiple statements for first and third expressions:
arr4[3]: 2 , arr4[3]: 2
arr4[4]: 3 , arr4[4]: 4
For loop without body:
arr5[1] :2
arr5[2] :4
```

- **Explanation of the code segment:** As stated in the explanation, the first for statement indicates to us that the zero value is considered as false in the conditional part of the for loop. Since the 0th element of arr is 0, it is not printed. The second for statement shows us the flexibility of optional

statements in for loop. The third for loop shows us the flexibility that multiple statements can be used in for statements. The last for loop also shows the flexibility of for statements without a loop body. All of these iterations are made on array data structure here.

- **Code Segment for WHILE LOOP and DO-WHILE LOOP:**

```
printf("While and do while loops: \n");
printf("0 value is considered as false:  \n");
int arr6[5] = {0,1,2,3,4};
i = 4;
while(arr6[i])
{
    printf("arr6[%d]: %d\n", i,arr6[i]);
    i--;
}
printf("while statement without body\n");
while(arr6[i]);
printf("\n");
printf("while loop vs do while loop\n");
while(arr6[i])
{
    printf("printed in while\n");
}

do{
    printf("printedin do-while\n");
} while(arr6[i]);
```

- **Result of Execution:**

```
While and do while loops:
0 value is considered as false:
arr6[4]: 4
arr6[3]: 3
arr6[2]: 2
arr6[1]: 1
while statement without body

while loop vs do while loop
printedin do-while
```

- **Explanation of the code segment:** First while loop executes the same output with the first for loop in the previous code segment. It shows us that we can simulate for loop logic in while loops. Also it indicates that the 0 values are

considered as false in the control statements of the while. Second while loop shows us that the while loop without body is valid in syntax in C language. Third while loop shows the difference between while and do-while loop. While loop first executes the control statement. Since it corresponds to 0, it does not execute the loop body. However, do-while first executes the body of the loop and then executes the control statement. Since it printed its body at least one as seen as the result of the execution.

2. **Data structures suitable for iteration**

- **Explanation:** Arrays are data structures that are suitable for iteration in C language.
- **Code Segment for Arrays:**

```
printf("Iteration of arrays:\n");
int array[] = {10,34,2,8,65,20};
int size = 6;
printf("Iteration with for:\n ");
for(i = 0; i < size ; i++)
{
    printf("array[%d] : %d\n", i, array[i]);

}
printf("Iteration with while:\n ");
i = 0;
while(i < size)
{
    printf("array[%d] : %d\n", i, array[i]);
    i++;
}
 printf("Iteration with do while:\n ");
   i = 0;
do
{
    printf("array[%d] : %d\n", i, array[i]);
    i++;
}while(i < size);
```

- **Result of Execution:**

```
Iteration of arrays:
Iteration with for:
 array[0] : 10
array[1] : 34
array[2] : 2
array[3] : 8
array[4] : 65
```

```
array[5] : 20
Iteration with while:
 array[0] : 10
array[1] : 34
array[2] : 2
array[3] : 8
array[4] : 65
array[5] : 20
Iteration with do while:
 array[0] : 10
array[1] : 34
array[2] : 2
array[3] : 8
array[4] : 65
array[5] : 20
```

- **Explanation of the code segment:** Arrays are data structures that are suitable for iteration in C language. We can iterate them by using for, while and do while loops as shown in the example code above.

3. **The way the next item is accessed**

- **Explanation:** All items in arrays are accessed through indexes. So, next item is accessed in C arrays through writing index "current index + 1".
- **Code Segment :**

```c
int array2 [] = {1,3,5,7};
size = 4;
int currentIndex = 0;
int currentNum = array2[currentIndex];
printf("current num: %d\n",currentNum);
int nextNum = array2[++currentIndex];
printf("next num: %d\n",nextNum);
nextNum = array2[++currentIndex];
 printf("next num: %d\n",nextNum);
nextNum = array2[++currentIndex];
 printf("next num: %d\n\n",nextNum);

 for(currentIndex = 0; currentIndex < size; currentIndex ++)
 {
     printf("next num: %d\n",array2[currentIndex]);
 }
```

- **Result of Execution:**

```
current num: 1
next num: 3
next num: 5
next num: 7

next num: 1
```

```
next num: 3
next num: 5
next num: 7
```

- **Explanation of the code segment:** We can access the next element in arrays in C language by using indexing. As seen from the example code, every time we want to access the next element in C, we increment the currentIndex by 1 In this way we access the next element. When iterating the array we do the same incrementation and accessing the element by indexing.

  My Evaluation about language: In my opinion, C does not have enough built-in iterable data structure. It only has arrays and it is too general. Besides that, all of the iteration statements (for, while and do while) make this language more writable and readable since they enable different iteration conveniences. However, some syntax details about them make it a little bit hard to remember. For example, remembering if we do not write the second expression in for loop, it is considered as true always and causes infinite loop.

GO

1. Iteration statements provided

- **Explanation:** GO language only provides for loops. For loops are **pretest-controlled** means they execute control statements before executing the body of the loop. In this language, for loops can be used in different forms. These are:
  - Simple for loop
  - Infinite loop
  - As While loop
  - Ranging Loop
- **Simple for loop**

  It has a syntax like the following:

  for initialization; condition; post{

  }

Its logic is similar to the other programming languages. This use corresponds to a counter-controlled loop. Initialization is optional and executed once. Condition corresponds to the boolean value and checks every iteration of the loop. Finally, post statements are executed after every iteration.

- **Infinite for loop**

  When all of the three statements are absent in the code, for loop becomes an infinite loop. Syntax like that:

  ```
  for{

        //statements

  }
  ```

- **For Loop as while loop**

  A for loop can simulate while loop. In this situation, we only write conditional statements to the for loop statement. Until the given condition will not be satisfied, for loop will execute its body. It can be considered as **logically-controlled** loops. Repetition is based on boolean expressions rather than counter.

- **Ranging Loop**

  It has the following syntax:

  ```
  for i, j:= range variable{

    // statement..
  }
  ```

  Where i and j are iteration variables. i iterates the variable and j holds the value of the elements of each iteration. It is a useful feature of for loop in terms of iterable data structures.
- **Code Segment for FOR LOOP :**

```
fmt.Println("Simple For Loop:")
array := [5] int {10,20,30,40,50}
for i := 0; i <5; i++{
    fmt.Println("array[",i,"]",array[i])
}

fmt.Println("Initialization is absent:")
j:= 0
 for  ;j <5; j++{
    fmt.Println("array[",j,"]",array[j])
}
fmt.Println("For Loop as while loop:")
array2 := [5] int {8,10,7,12,15}
i := 0
for array2[i] < 11{
    fmt.Println("array2[",i,"]",array2[i])
    i++
}
fmt.Println("Ranging for loop:")
array3 := [4] int {12,13,50,26}
for k, j:= range array3{
     fmt.Println("array3[",k,"]",j)
}
```

- **Result of Execution:**

```
Simple For Loop:
array[ 0 ] 10
array[ 1 ] 20
array[ 2 ] 30
array[ 3 ] 40
array[ 4 ] 50
Initialization is absent:
array[ 0 ] 10
array[ 1 ] 20
array[ 2 ] 30
array[ 3 ] 40
array[ 4 ] 50
For Loop as while loop:
array2[ 0 ] 8
array2[ 1 ] 10
array2[ 2 ] 7
Ranging for loop:
array3[ 0 ] 12
```

```
array3[ 1 ] 13
array3[ 2 ] 50
array3[ 3 ] 26
```

- **Explanation of the code segment:** First for loop shows the simple for loop that we are familiar with other programming languages. It iterates through an array. Second loop shows us that in the simple counter-controlled loop we do not have to write an initialization statement.Third for loop is for loop as a while loop. It only takes conditional statements and continues to execute the loop body until the statement "array2[i] < 11" holds true. It is not a counter-controlled for loop sample. It is logically-controlled. Third one is a syntax of for loop that iterates by using k value which ranges from the beginning to the end of the array3.

2. Data structures suitable for iteration

- **Explanation:** By using the ranging loop, we can iterate data structures such as arrays, strings,maps and channels easily in GO language. Arrays can be iterated by using for loops as shown in the previous code segment.
- **Code Segment for Strings :**

```
fmt.Println("Simple For Loop:")
str := "Ilke"
 for i := 0; i <4; i++{
    fmt.Printf("Index number: %U, %d, char: %c\n",str[i],i,str[i])
}
fmt.Println("For Loop as while loop:")
str2 := "Love"
indx := 0
for indx < len(str2){
    fmt.Printf("Index number: %U, %d, char: %c\n",str2[indx],indx,str2[indx])
    indx++
}
fmt.Println("Ranging for loop:")
 for k, j:= range str2{
      fmt.Printf("Index number: %U, %d, char: %c\n",j,k,j)
}
```

- **Result of Execution:**

```
Simple For Loop:
Index number: U+0049, 0, char: I
Index number: U+006C, 1, char: l
Index number: U+006B, 2, char: k
Index number: U+0065, 3, char: e
For Loop as while loop:
Index number: U+004C, 0, char: L
Index number: U+006F, 1, char: o
```

```
Index number: U+0076, 2, char: v
Index number: U+0065, 3, char: e
Ranging for loop:
Index number: U+004C, 0, char: L
Index number: U+006F, 1, char: o
Index number: U+0076, 2, char: v
Index number: U+0065, 3, char: e
```

- **Explanation of the code segment:** First for loop iterates string with simple for loop logic (counter controlled). Second for loop also iterates string by using for loop as while by only look for condition i <len(str2). However it can still be considered as a counter-controlled loop since the index counter is used in this loop. Finally, ranging for loop is used.

- **Code Segment for Maps and Channels :**

```
fmt.Println("Iteration of Maps")
var name_map = map[int] string{
    1: "İlke",
    2: "Aysu",
    3: "Zeynep",
}
 fmt.Println("Ranging for loop:")
 for key,name := range name_map{
     fmt.Println("key: ",key, "name: ", name)
 }
 fmt.Println("Iteration of Channels")
 channel := make(chan int,4)
    channel<- 1
    channel<- 20
    channel <- 12
    channel <- 100
    close(channel)
 for i:= range channel{
     fmt.Println(i)
 }
```

- **Result of Execution:**

```
Iteration of Maps
Ranging for loop:
key:  1 name:  İlke
```

```
key:  2 name:  Aysu
key:  3 name:  Zeynep
Iteration of Channels
1
20
12
100
```

- **Explanation of the code segment:** A map in GO language is a key-value pair storage. It is an associative array. A loop that ranges the values of a map can be used as an iterator in GO language. Channel is another data structure that is suitable for iteration. Again for loops that range through the elements of the channel can be used to iterate channel data structure.

3. The way the next item is accessed

- **Explanation:** All items in arrays are accessed through indexes. So, the next item is accessed in GO arrays through writing index "current index + 1". Besides, strings characters can be accessed through the same manner of the arrays. However, there is no manual way to access the "next" element of maps without knowing the key values in GO since they do not have indexes or they are not ordered. Without knowing the key values one cannot access the next item of a map directly. However, it is possible to iterate maps through ranging loops in GO languages. One can access the next element in this way. Finally, It is possible to access the next values of channels by using again ranging for loops. Also, one can access the next element in channel manually, however this cause that this element is deleted from the channel.
- **Code Segment for Arrays and Strings:**

```
arr := [4] int {1,3,5,7}
size2 := 4
currentIndex := 0
currentNum := arr[currentIndex]
fmt.Println("current num: ",currentNum)
nextNum:= arr[currentIndex +1]
fmt.Println("next num: ", nextNum)
currentIndex++
nextNum2:= arr[currentIndex +1]
fmt.Println("next num: ", nextNum2)
currentIndex++
nextNum3:= arr[currentIndex +1]
fmt.Println("next num: ", nextNum3)

for currentIndex:= 0; currentIndex <size2; currentIndex++{
    fmt.Println("next num:", arr[currentIndex])
}
```

```
fmt.Println("For strings")
currentIndex2 := 0
str3 := "Ilke"
fmt.Printf("str3[%d]:%c\n",currentIndex2, str3[currentIndex2])
currentIndex2++
fmt.Printf("str3[%d]:%c\n",currentIndex2, str3[currentIndex2])
```

- **Result of Execution:**

```
For arrays
current num:  1
next num:  3
next num:  5
next num:  7
next num: 1
next num: 3
next num: 5
next num: 7
For strings
str3[0]:I
str3[1]:l
```

- **Explanation of the code segment:** We can access the next element in arrays in GO language by using indexing. As seen from the example code, every time we want to access the next element in GO, we increment the currentIndex by 1 In this way we access the next element. When iterating the array we do the same incrementation and accessing the element by indexing. In same manner, we can access the characters of the string by incrementing the current index. We do similar logic when iterating strings and arrays by using for loops.

- **Code Segment for Channels :**

```
fmt.Println("Accessing next element of Channels")
    channel2:= make(chan int,4)
    channel2 <- 1
    channel2 <- 2
    channel2 <- 3
    channel2 <- 4
    element := <- channel2
   fmt.Println("element: ",element)
   element2 := <- channel2
   close(channel2)
   fmt.Println("element: ",element2)
    for i := range channel2{
        fmt.Println(i)
    }
      for k := range channel2{
        fmt.Println(k)
    }
```

- **Result of Execution:**

```
Accessing next element of Channels
element:  1
element:  2
3
4
```

- **Explanation of the code segment:**

  Since Maps are associative arrays, it is impossible to access “next” elements without knowing the key value by hand. However, as we showed previously it is possible to iterate the maps by using ranging for loops. Besides, next elements can also be accessed in channels, again by using ranging for loops. Also, one can access the next element of the channel one by one by hand. However, as seen from the result of the code, iterating channel is different than the others. In every access, it deletes the value from the channel. We can understand this by looking at the second loop. It does not print any value since chanell is nil.

  My Evaluation about language: Since it has more data structures that are suitable to iteration, it seems more writable. However, the details of these structures are hard to remember. So, it makes this language less readable and writable.

Javascript

1. Iteration statements provided

- **Explanation:**There are three types of iteration statements in C language. These are:
  - for loop
  - while loop
  - do... while loop
  - labels
  - **For loop:**

    Its syntax like in the following:

    for(expression1; expression2 ; expression3)

        loop body

    Loop body can be a single, compound or a null statement. Expression1 is executed once and it generally holds initialization of the counter in JavaScript. Expression2 is executed before each iteration and it generally

holds for the conditions. Expression 3 is executed at the end of each iteration, it holds the update statement for counters generally such as incrementation or decrementation. Initial expression is optional. To execute multiple statements we will use {} block statements.

- **DO-While loop:**

  Its syntax like in the following:,

  do

  loop body

  while (control_expression);

  Do while statements are **posttest logical-controlled loops.** In do-while, the loop body is executed until the control expression evaluates to false. This means at least once the body of the loop will be executed.

- **While loop:**

  Its syntax like in the following:

  while (control_expression)

  loop body

  While statements can simulate for loop logic. While statements are **pretest logical-controlled loops.** So, we cannot always know the iteration number of the loop. It executes the body as long as the conditional expression is true.

- **Code Segment for FOR :**

```
document.write("For loops<br\>");
let arr = [0,1,2,3,4];
for(let i = 0; i< arr.length; i ++)
{
     document.write("arr[",i,"]",arr[i],"<br\>");
}
 document.write("First and last expression are absent<br\>");
 let i = 0;
   for(; i< arr.length;)
{
     document.write("arr[",i,"]",arr[i],"<br\>");
     i++;
}
document.write("For loop without body<br\>");
let j = 0;
 for(; j< arr.length;j++);
  document.write("j:",j,"<br\>");
```

- **Result of Execution:**

For loops
arr[0]0
arr[1]1
arr[2]2
arr[3]3
arr[4]4
First and last expression are absent
arr[0]0
arr[1]1
arr[2]2
arr[3]3
arr[4]4
For loop without body
j:5

- **Explanation of the code segment:** The first for statement indicates to us the syntax of the for loop in JavaScript. The second for statement shows us the flexibility of optional statements in for loop. The last for loop also shows the flexibility of for statements without a loop body. All of these iterations are made on array data structure here.

- **Code Segment for WHILE and DO-WHILE :**

```
document.write("While and do while loops<br\>");
j =0;
arr =[5,6,7,8];
while(j<arr.length)
{
   document.write("arr[",j,"]",arr[j],"<br\>");
   j++;
}
document.write("While loop without body<br\>");
while(j<0);
document.write("While loop vs. do while loop<br\>");
j = 0;
while(j)
{
  document.write("Print in while<br\>");
}
do
{
  document.write("Print in do-while<br\>");
}while(j);
```

- **Result of Execution:**

While and do while loops
arr[0]5
arr[1]6
arr[2]7
arr[3]8
While loop without body
While loop vs. do while loop
Print in do-while

- **Explanation of the code segment:**First while loop executes the same output with the first for loop in the previous code segment. It shows us that we can simulate for loop logic in while loops. Second while loop shows us that the while loop without body is valid in syntax in JavaScript language. Third while loop shows the difference between while and do-while loop. While loop first executes the control statement. Since it corresponds to 0, it does not execute the loop body. However, do-while first executes the body of the loop and then

executes the control statement. Since it printed its body at least one as seen as the result of the execution.

2. Data structures suitable for iteration

- **Explanation:**We can iterate data structures such as arrays, strings and objects easily in GO language. Arrays can be iterated by using for loops as shown in the previous code segment.
- **Code Segment for Strings :**

```
 document.write("Strings<br\>");
 document.write("For loops<br\>");
let str = "Ilke";
   for(let i = 0; i< str.length; i ++)
   {
        document.write("str[",i,"]: ",str[i],"<br\>");
   }
   document.write("While loops<br\>");
   j = 0;
   while(j<arr.length)
    {
        document.write("str[",j,"]",str[j],"<br\>");
       j++;
   }
   document.write("Do while loops<br\>");
   j = 0;
    do
{
   document.write("str[",j,"]",str[j],"<br\>");
   j++;
}while(j < str.length);
```

- **Result of Execution:**

  Strings
  For loops
  str[0]: I
  str[1]: l
  str[2]: k
  str[3]: e
  While loops
  str[0]I
  str[1]l
  str[2]k
  str[3]e
  Do while loops
  str[0]I
  str[1]l
  str[2]k
  str[3]e

- **Explanation of the code segment:**First for loop iterates string with simple for loop logic (counter controlled). In there, we iterate the string by using index and by incrementing this index, we access the next element.
- **Code Segment for Objects :**

```
document.write("Objects<br\>");
document.write("For loops<br\>");
var book = {name: "Faust", author: "Goethe", year: "1806"};
for(let key in book)
{
  document.write(key," : ", book[key],"<br\>");
}
```

- **Result of Execution:**

  Objects
  For loops
  name : Faust
  author : Goethe
  year : 1806

- **Explanation of the code segment:** Objects in JavaScript contain name:value pairs. It is possible to iterate them by using for loop in the form of for(let key in book) as in the example. Since it is not an ordered container, we cannot just

basically traverse the indexes of the objects to iterate it. We need to know key values. “for(let key in book)” statement enable them to us.

3. The way the next item is accessed

- **Explanation:**All items in arrays are accessed through indexes. So, the next item is accessed in JavaScript arrays through writing index “current index + 1”. Besides, strings characters can be accessed through the same manner of the arrays. However, there is no manual way to access the “next” element of objects without knowing the key values in JavaScript since they do not have indexes or they are not ordered. Without knowing the key values one cannot access the next item of an object directly. However, it is possible to iterate objects through for loops in JavaScript languages. One can access the next element in this way.
- **Code Segment for :**

```
document.write("For arrays<br\>");
var array  = [1,3,5,7];
let curIndx = 0;
document.write("current number: ", array[curIndx],"<br\>");
document.write("next number: ", array[curIndx +1], "<br\>");
 document.write("For strings<br\>");
str  = "Love";
curIndx = 0;
document.write("current character: ", str[curIndx],"<br\>");
document.write("next character : ", str[curIndx +1], "<br\>");
```

- **Result of Execution:**

For arrays
current number: 1
next number: 3
For strings
current character: L
next character : o

- **Explanation of the code segment:** In arrays and strings, it is possible to access the next elements thanks to indexing. By using the “curIndx +1” we can access the next item in these two data structures. However, without knowing the key values, we cannot get the elements of the objects in JavaScript since objects ar not ordered or indexed data structures. However, if we want by using Object.keys() we can get the key values of the object and iterate according to these key values. In addition to these, as we mentioned previously, we can use “for(let key in book)” form of the for statement to get the next elements and iterate on the object data structures.

My Evaluation about language: I think JavaScript is better than C and GO in terms of iteration statements. It contains more data structures that are suitable for iteration than C language. Also, these structures are less complicated to use than the GO language (channels).

PHP

1. Iteration statements provided

- **Explanation:**There are four types of iteration statements in PHP language. These are:
    - for loop
    - while loop
    - do... while loop
    - foreach loop

      for, while and do while loops have the same logic with other programming languages. foreach loops works only for arrays. They are used to iterate through each key: value pairs in an array. Besides these, there are predefined iterators in PHP.**current, next,prev** and **reset** words are used on arrays to iterate. PHP has iterable values which can be looped through with a foreach loop.

- **Code Segment for FOR, WHILE,DO WHILE :**

```php
echo "---------1----------</br>";
echo "For Loops </br>";
$arr = array(0,1,2,3,4);
for($x = 0 ; $x < sizeof($arr); $x++)
{
    echo "arr[",$x,"]: ",$arr[$x],"</br>";
}
$x = 0;
echo "First and last expressions are absent</br>";
 for( ; $x < sizeof($arr); )
{
    echo "arr[",$x,"]: ",$arr[$x],"</br>";
    $x++;
}
$i = 0;
echo "For loop without body</br>";
 for( ; $i < sizeof($arr); $i++ );
 echo "i: ",$i,"</br>";
echo "While and do while loops</br>";
$j = 0;
while($j < sizeof($arr))
{
    echo "arr[",$j,"]: ",$arr[$j],"</br>";
    $j++;
}
echo "While loop without body</br>";
while($j < 0);
echo "While loop vs. do while loop</br>";
while($arr[0])
{
    echo "Print in while</br>";
}
```

```
do
{
    echo "Print in do while</br>";
}while($arr[0]);
```

- **Result of Execution:**

```
---------1----------
For Loops
arr[0]: 0
arr[1]: 1
arr[2]: 2
arr[3]: 3
arr[4]: 4
First and last expressions are absent
arr[0]: 0
arr[1]: 1
arr[2]: 2
arr[3]: 3
arr[4]: 4
For loop without body
i: 5
While and do while loops
arr[0]: 0
arr[1]: 1
arr[2]: 2
arr[3]: 3
arr[4]: 4
While loop without body
While loop vs. do while loop
Print in do while
```

- **Explanation of the code segment:**The first for statement indicates to us the syntax of the for loop in PHP. The second for statement shows us the flexibility of optional statements in for loop. The last for loop also shows the flexibility of for statements without a loop body. All of these iterations are made on array data structure here.

First while loop executes the same output with the first for loop. It shows us that we can simulate for loop logic in while loops. Second while loop shows us that the while loop without body is valid in syntax in PHP language. Third while loop shows the difference between while and do-while loop. While loop first executes the control statement. Since it corresponds to 0, it does not execute the loop body. However, do-while first executes the body of the loop and then executes the control statement. Since it printed its body at least one as seen as the result of the execution.

- **Code Segment for FOR EACH:**

```
echo "For Each Loops </br>";
$list = array("milk","bread","egg","tomato");
foreach($list as $item)
{
    echo "$item </br>";
}
$person_list = array("İlke" => "milk", "Zeynep" => "bread", "Aysu" => "egg");
foreach($person_list as $person => $item)
{
    echo "$person => $item </br>";
}
```

- **Result of Execution:**

For Each Loops
milk
bread
egg
tomato
İlke => milk
Zeynep => bread
Aysu => egg

- **Explanation of the code segment:** In the first foreach loop, in every iteration $item value is assigned to the current element at the $list. In this way, we can traverse arrays without using any index value.In the second foreach loop, we iterate associative arrays. They are also iterable.

2. Data structures suitable for iteration

- **Explanation:** In PHP, iterable is defined as any value that can be iterable through a foreach loop. This keyword can be used as data type. All arrays are iterables by default in PHP language. So, arrays are suitable data structures for iteration as many other languages. We mentioned foreach examples of arrays previously. Besides these loops, predefined iterators are used for iterative

access to PHP's arrays. They use **next, prev, current.** Besides numeric arrays, in PHP there are also associative arrays and they are also suitable for iteration with foreach as we show previously.

- **Code Segment for String :**

```
echo "Strings</br>";
$str = "Ilke";
for($i = 0; $i <strlen($str); $i++)
{
    echo "$str[$i] </br>";
}
```

- **Result of Execution:**

Strings
I
l
k
e

- **Explanation of the code segment:**First for loop iterates string with simple for loop logic (counter controlled). In there, we iterate the string by using index and by incrementing this index, we access the next element. However, it is not possible to iterate strings by using foreach loop. To do that first, we need to convert the string to array and iterate on this array by using foreach.

- **Code Segment for predefined iterators:**

```
$numbers = array(1,3,5,7,9);
echo "cur:",current($numbers). "</br>";
echo "next: ",next($numbers)."</br>";
echo "cur: ",current($numbers). "</br>";
echo "next: ",next($numbers)."</br>";
echo "cur: ",current($numbers). "</br>";
echo "prev: ",prev($numbers)."</br>";
echo "cur: ",current($numbers). "</br>";
echo "cur:",reset($numbers). "</br>";
while($cur_val = next($numbers))
    echo "next num: ", $cur_val."</br>";
```

- **Result of Execution:**

```
cur:1
next: 3
cur: 3
next: 5
cur: 5
prev: 3
cur: 3
cur:1
next num: 3
next num: 5
next num: 7
next num: 9
```

- **Explanation of the code segment:** The current pointer points to the internal pointer that accesses the element in the array. The next iterator moves the current pointer to the next element. The prev iterator moves the current pointer to the previous element. Finally reset will move the current pointer to the first element of the array. We can iterate the array with these iterator functions.

3. The way the next item is accessed

- **Explanation:**All items in arrays are accessed through indexes. So, the next item is accessed in PHP arrays through writing index “current index + 1”. Besides, strings characters can be accessed through the same manner of the arrays. There is a way to access the “next” element of associative arrays without knowing the key values in PHP unlike other languages. We can use predefined iteratives next(), prev() on associative arrays.Besides, it is possible to iterate associative arrays through for loops in PHP languages. One can access the next element in this way. Besides these, as we mentioned in the previous code, there are predefined iterators for arrays in PHP. One can access the next item by using **next()** function on arrays.

- **Code Segment for:**

```php
$numbers = array(1,3,5,7,9);
echo "cur:",current($numbers). "</br>";
echo "next: ",next($numbers)."</br>";
echo "cur: ",current($numbers). "</br>";
echo "next: ",next($numbers)."</br>";
echo "cur: ",current($numbers). "</br>";
echo "prev: ",prev($numbers)."</br>";
echo "cur: ",current($numbers). "</br>";
echo "cur:",reset($numbers). "</br>";
while($cur_val = next($numbers))
    echo "next num: ", $cur_val."</br>";

$person_list2 = array("İlke" => "milk", "Zeynep" => "bread", "Aysu" => "egg");
echo "cur:",current($person_list2). "</br>";
echo "next: ",next($person_list2)."</br>";
echo "cur: ",current($person_list2). "</br>";
echo "next: ",next($person_list2)."</br>";
echo "cur: ",current($person_list2). "</br>";
echo "prev: ",prev($person_list2)."</br>";
echo "cur: ",current($person_list2). "</br>";
echo "cur:",reset($person_list2). "</br>";
while($cur_val = next($person_list2))
    echo "next num: ", $cur_val."</br>";
```

- **Result of Execution:**

cur:1
next: 3
cur: 3
next: 5
cur: 5
prev: 3
cur: 3
cur:1
next num: 3
next num: 5
next num: 7
next num: 9
cur:milk
next: bread
cur: bread
next: egg
cur: egg
prev: bread
cur: bread
cur:milk
next num: bread
next num: egg

- **Explanation of the code segment:** Besides numeric arrays, associative arrays can be iterated through predefined iterators. Next element acces can be done in associative arrays by using next() function that moves the internal current pointer to the next element.

  My Evaluation about language: Since PHP has many ways to iterate data structures such as for, foreach,while,do-while,predefined iterators, it is writable. For the person who knows the language very well, this is a good feature. However, remembering the details of these may detriment the readability and writability and make it hard to learn language for a beginner person. In many languages, there is not so much a way as PHP.

Python

1. Iteration statements provided

- **Explanation:** There are two types of loop in Python. These are:
  - for loop
  - while loop

  For loops:

  For loop in python is used to iterate over a sequence. This sequence can be a list, string, tuple, dictionary or a range. Its syntax like in the following:

  for x in container:

  statement

  or

  for x in range(y):

  statement

  While loops:

  It executes the statements as long as the condition is true. Its syntax like in the following:

while control_statement:

statements

- **Code Segment for FOR and WHILE:**

```
print("For loops")
list1 = [0,1,2,3,4]
for x in range(len(list1)):
    print("list1[",x,"]: ", list1[x])
for x in list1:
    print(x)
print("While loops")
i = 0
while i <len(list1):
    print("list1[",i,"]: ", list1[i])
    i = i +1
print("Nested loops")
list2 = [3,45,67]
for x in list2:
    for y in list2:
        if y > x:
            print(y)
```

- **Result of Execution:**

```
For loops
list1[ 0 ]:  0
list1[ 1 ]:  1
list1[ 2 ]:  2
list1[ 3 ]:  3
list1[ 4 ]:  4
0
1
2
3
4
While loops
list1[ 0 ]:  0
list1[ 1 ]:  1
list1[ 2 ]:  2
list1[ 3 ]:  3
list1[ 4 ]:  4
Nested loops
45
67
67
```

- **Explanation of the code segment:** First loop that is used in the above code will execute the for loop as in the same logic as array indexing in other languages. However, there are no arrays in python, instead there are lists. Besides that, there is an easier way rather than using indexes. Second loop shows the easier way. While loop is used in the same ways in other programming languages. Besides these, like other languages, nested loops are supported in python.

2. Data structures suitable for iteration

- **Explanation:** There are many data structures suitable for iteration in Python language. These are lists, dictionaries, tuples, strings, and strings.
- **Code Segment:**

```
print("Iteration over List")
list3 =[3,4,7]
for x in list3:
    print(x)
print("Iteration over tuple")
tuple1 = (3,4,7)
for x in tuple1:
    print(x)
print("Iteration over dictionary")
dict1 = {"İlke": 20, "Ali": 12}
for x in dict1:
    print(x,":", dict1[x])
print("Iteration over set")
set1 = {3,4,5}
for x in dict1:
    print(x)
str = "hello"
print("Iteration over string")
for x in str:
    print(x)
```

- **Result of Execution:**

```
Iteration over List
3
4
7
Iteration over tuple
3
4
7
Iteration over dictionary
İlke : 20
Ali : 12
Iteration over set
İlke
Ali
Iteration over string
h
e
l
l
o
```

- **Explanation of the code segment:** All of the types in the code lists, tuples, strings, dictionaries and sets are iterable data structures.

3. The way the next item is accessed

- **Explanation:** Besides loops, it is possible to create iterators on python language. These iterators can be used on lists, tuples, sets and strings. They cannot be used in dictionaries.
- **Code Segment for Iterator :**

```
print("Iterator over list")
iterList = iter([5,9,8])
x = next(iterList)
print(x)
x = next(iterList)
print(x)
x = next(iterList)
print(x)
print("Iterator over tuple")
iterTuple = iter((5,9,8))
x = next(iterTuple)
print(x)
x = next(iterTuple)
print(x)
x = next(iterTuple)
print(x)
print("Iterator over set")
iterSet = iter( {3,4,5})
x = next(iterSet)
print(x)
x = next(iterSet)
print(x)
x = next(iterSet)
print(x)
print("Iterator over string")
iterStr = iter( "Ilk")
x = next(iterStr)
print(x)
x = next(iterStr)
print(x)
x = next(iterStr)
print(x)
```

- **Result of Execution:**

```
Iterator over list
5
9
8
Iterator over tuple
5
9
8
Iterator over set
3
4
5
Iterator over string
I
l
k
```

- **Explanation of the code segment:**

  My Evaluation about language: I think Python is the best language in terms of iteration of the statements. It has many data structures that are suitable for iteration but these structures are not so complex as GO. Besides it does not have complex loop features and enable iterator and next() function for the iterations. Since it is not so complex in terms of both data structures and iteration ways, it is readable and writable.

Rust

1. Iteration statements provided

- **Explanation:** There are three type of loops that are provided by Rust language. These are:
  - for
  - while
  - loop

For Loop:

Its execution logic is the same as other programming languages. Its syntax like in the following:

```
for variable in lower_bound..upper_bound{

    statements

}
```

While Loop:

Its execution logic is the same as other programming languages. Its syntax like in the following:

```
while control_stmt{

    statements

}
```

Loop:

It iterates until the **break** statement is executed. The control of the loop is placed in the if statement in the loop. Its body is executed at least once. Therefore, it is similar to the do-while structure.Its syntax like in the following:

```
loop{

    statements

    if conditional_stmt{

        break;

    }

}
```

- **Code Segment:**

```
println!("-------1----");
let mut array: [i32;3] =[0;3];
array[0] = 5;
array[1] = 9;
array[2] = 13;
println!("For loops: ");
for temp in &array{
    println!("temp is {}",temp);
}
 println!("While loops: ");
let mut i = 0;
while i < array.len()
{
    println!("array[{}] : {}",i,array[i]);
   i += 1;

}
println!(" Loops: ");
let mut j = 0;
loop{
     println!("array[{}] : {}",j,array[j]);
     j+=1;

     if j == array.len(){
         break;
     }
}
```

- **Result of Execution:**

```
-------1----
For loops:
temp is 5
temp is 9
temp is 13
While loops:
array[0] : 5
array[1] : 9
array[2] : 13
 Loops:
array[0] : 5
array[1] : 9
array[2] : 13
```

- **Explanation of the code segment:** We can iterate array by using for loop without using any indexing as in the example. While and loop in the code use indexing to iterate the array.

2. Data structures suitable for iteration

- **Explanation:** Arrays are suitable data structures for iteration in Rust as the other languages as we mentioned in the previous code. Besides, vectors can be iterated through for, while and loop.
- **Code Segment:**

```
println!("------------2---------");
println!("For loops: ");
let v = vec![10,20,30,40];
let iter_v = v.iter();
for temp in iter_v{
    println!("temp is {}",temp);
}
println!("While loops: ");
let mut k = 0;
while k < v.len()
{
    println!("v[{}] : {}",k,v[k]);
   k += 1;

}
println!(" Loops: ");
let mut m = 0;
loop{
     println!("v[{}] : {}",m,v[m]);
     m+=1;

     if m == v.len(){
         break;
     }
}
```

- **Result of Execution:**

```
------------2---------
For loops:
temp is 10
temp is 20
temp is 30
temp is 40
While loops:
v[0] : 10
v[1] : 20
v[2] : 30
v[3] : 40
 Loops:
v[0] : 10
v[1] : 20
v[2] : 30
v[3] : 40
```

3. The way the next item is accessed

- **Explanation:** There is a predefined iterator in the Rust language. After defined in the code, it is possible to use it by **next()** function.
- **Code Segment:**

```
println!("------------3---------");
let mut array2: [i32;3] =[0;3];
array2[0] = 15;
array2[1] = 29;
array2[2] = 33;
let mut arriter = array2.iter();
println!("{:?}",arriter.next());
println!("{:?}",arriter.next());

let mut array3: [i32;3] =[0;3];
array3[0] = 15;
array3[1] = 29;
array3[2] = 33;
let  arriter3 = array3.iter();
for temp in arriter3{
    println!("temp is {}", temp);
}

//not valid
//let v2 = vec![10,20,30,40];
//let iter_v = v2.iter();
//println!("{:?}",v2.next());
//println!("{:?}",v2.next());
```

- **Result of Execution:**

```
------------3---------
Some(15)
Some(29)
temp is 15
temp is 29
temp is 33
```

- **Explanation of the code segment:** In the code, we use an iterator on arrays. Even if the iterators can be used in Rust, as we used in the previous code segment, it is not valid to use next() function on vectors in Rust.

  My Evaluation about language: I think the Rust language is hard to read and write according to the syntax of its data structures which are iterable such as arrays. In terms of iteration statements, it uses the similar logic as the other languages.

Conclusion

1. Which language is the best in terms of iterations and why?

I think Python is the best language in terms of iteration of the statements. It has many data structures that are suitable for iteration but these structures are not so complex as GO and Rust. Besides it does not have complex loop features and it enables an iterator and next() function for the iterations. Since it is not so complex in terms of both data structures and iteration ways, it is readable and writable.

2. My learning Strategy

I learn these languages by using trial-and-error methods by using online compilers.

-Materials and tools I used

I searched a lot of internet sources to understand the syntax of the programming languages and I also used our textbook to find some information. Some of these sites are:

For GO:

Loops in Go Language - GeeksforGeeks
https://golangdocs.com/maps-in-golang
https://www.golangprograms.com/go-language/arrays.html
https://golangbot.com/channels/
https://www.golangprograms.com/how-to-iterate-over-a-map-using-for-loop-in-go.html

For PHP:

https://www.w3schools.com/php/php_looping_foreach.asp
https://www.w3schools.com/Php/func_array_reset.asp
https://stackoverflow.com/questions/4601032/php-iterate-on-string-characters
https://www.w3schools.com/Php/php_arrays_associative.asp
https://www.w3schools.com/php/php_iterables.asp

For JavaScript:

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Loops_and_iteration
https://www.javascripttutorial.net/object/iterate-object-in-javascript/
https://code-boxx.com/iterate-over-objects-javascript/
https://www.w3schools.com/js/js_arrays.asp

For Python:

https://www.w3schools.com/python/ref_func_next.asp
https://www.w3schools.com/python/python_for_loops.asp
https://www.w3schools.com/python/python_dictionaries.asp
https://www.w3schools.com/python/python_sets.asp

For Rust:

https://www.tutorialspoint.com/rust/rust_loop.htm

-Experiments I performed

I performed so many experiments by using compilers When I failed to understand the syntax and semantics, I searched for it. Besides that, I try to guess the results of these expressions and verify them by using an online compiler. For example, when I try to learn whether an iterator is valid or not on a specific data structure, I write it and try to compile it. If it does not compile, I search for other solutions for it. Addition to these, I tried to understand the valid syntax of the language about loops by testing. For example, loops without body, missing expressions etc.

-URL s of online compilers/interpreters

C:
https://www.onlinegdb.com/online_c_compiler
GO:
https://www.online-ide.com/online_golang_compiler
JS:
Tryit Editor v3.6 (w3schools.com)
PHP:
PHP Tryit Editor v1.2 (w3schools.com)
Python:
https://www.online-python.com/online_python_compiler
Rust:
https://www.tutorialspoint.com/compile_rust_online.php