Elif Gülşah Kaşdoğan

CS315- HW1

Section: 3

21601183

C:

1. C uses pointer arithmetic to calculate positions of the array items. arr[5] and 5[arr] can be used as equalivent however 5[arr] is found confusing for programmers. In short C supports pointers as subscripts but it is not widely used, integers are used as array subscripts.

```
-bash-4.2$ more myEx.c
#include <stdio.h>

main(){
  int array[4]={1,2,3,4};
  printf("%d\n",array[0]); //1
  printf("%d\n", 1[array]); //2
  printf("%d\n",array[8]); //garbage
}
-bash-4.2$ ./myEx
1
2
0
```

2. Array references are not range checked. C based languages do not perform boundary check when you want to access an array element because of the performance issues. Reaching a memory location which was not allocated for array is possible and it is responsibility of programmer to perform operation in proper bounds.
3. Ranges of subscript are bound in compile time. Arrays in C cannot grow or shrink directly after compilation.
4. Allocation takes place in compile time. Memory needed will be allocated before run time.
5. For ragged arrays compiler would not complain. Rectangular arrays are allowed. Uninitialized portion of ragged arrays are set to 0, in other words C creates a rectangular array for ragged arrays and fills the positions with values which may lead programmer to do mistakes. Both rectangular and ragged arrays allowed.

```
#include <stdio.h>
int main () {

  int a[2][3] = { {1,2,3}, {4,5,6}};//rectangular
  int i, j;
  int b[2][3] = { {1,2,3}, {5}};//ragged

   for ( i = 0; i < 2; i++ ) {
    for ( j = 0; j < 3; j++ ) {
      printf("a[%d][%d] = %d\n", i,j, a[i][j] );
```

```
 }
}

for( i = 0; i<2; i++){
  for(j = 0; j < 3; j++){
    printf("b[%d][%d] = %d\n", i, j, b[i][j]);
  }
}

 return 0;
}
-bash-4.2$ ./myEx
a[0][0] = 1
a[0][1] = 2
a[0][2] = 3
a[1][0] = 4
a[1][1] = 5
a[1][2] = 6
b[0][0] = 1
b[0][1] = 2
b[0][2] = 3
b[1][0] = 5
b[1][1] = 0
b[1][2] = 0
```

6. Index of the last element in array gives us maximum subscript. maximum subscript will be size-1 for C arrays.

   ```
   int size = sizeof(array)/sizeof(int); // the one used previously {1,2,3,4}
   printf("This is my last element %d and maximum subscript %d\n", array[size-1], size-1);
   ```

   sizeof returns the bytes consumed for storing this array, size of int depends on architecture of computer. If we divide total bytes consumed to bytes consumed for one int weget the size of the array, last elements index will be size-1.
7. Arrays are special data structures in C, array objects cannot be initialized in C. If by initialization it is meant that if I can perform an operation like int array[4]={1,2,3,4}; we can say that array items can be initialized like this.
8. C does not support slices.

Javascript:

1. Normally objects use named indexes in JS, however arrays are special kind of objects they use numbered indexes. Javascript does not support named indexes for arrays, if you use it may create incorrect result. In other words JS does not support associative arrays.

   ```
   var array1 = ["Gulsah", " Kasdogan", " CS"];
   ```

```
var array2 = ["Elif ", 20, 2016]; //elements can be different types

var array3 = {firstname: "Gulsah", lastname: "Kasdogan", age: 20, year: 2018};
```

However members of array3 can be accesed by names of variables. but if you try something like:

```
var student = [];
person["firstName"] = "Gulsah";
person["lastName"] = "Kasdogan";
person["id"] = 216;
```

You will obtain undefined results. array3 example will not be problematic but addition with named indexes is problematic. names and integers are legal subscripts in Javascript but named indexes must be used after initialization of array.

2. Javascript performs run-time bounds checking. If desired index is out of bounds it prints "undefined" on console and proceeds execution from where it is left.
3. Ranges of subscript are bound after declaretion and will be updated as new elements added.
4. Javascript does not define a memory layout; usage of memory will depend on implementation of the interpreter. Javascript has a garbage collector, so it uses heap for some specific memory allocations. Allocation takes place after declaration of array. Array can grow in runtime in JS, allocation takes place just in time in this case.
5. JS allows multidimensional arrays, they are in form array of arrays. Both rectangular and multidimensional arrays are allowed in JS. Example code: myjs2.htm

```
var rectangular = [
[1,2],
   [3,4]
];

var jagged = [
   ['a', 'b', 'c'],
   ['d', 'e', 'f'],
   ['g', 'h', 'j', 'k','e','k', 'z']
];
```

6. maximum number of subscripts: size-1
7. Array objects can be initialized as
   var array = new Array(item1, item2, item3..);

```
var arrayObject = new Array(5);

for(var i = 0; i<5;  i++){

        arrayObject[i] = i;
}
```

```
document.getElementById("demo").innerHTML = arrayObject;
```

8. Array slicing is supported in JS. slice() function returns the new array object with specified items.

```
var sliced  = array.slice(first, end);

var arrayObject = new Array(10);

for(var i = 0; i<arrayObject.length;  i++){

        arrayObject[i] = i+1;
}

var sliced = arrayObject.slice(3,6);
```

Python:

1. Integers are the only legal subscripts for python arrays.
2. Python uses run-time bound check.

```
array = ["Banana", "Apple", "Pear" , "Melon"]
for x in range(0, len(array)):
   print(array[x])

print(array[9])

-bash-4.2$ python3 my1.py
Banana
Apple
Pear
Melon
Traceback (most recent call last):
  File "my1.py", line 5, in <module>
    print(array[9])
IndexError: list index out of range
```

3. Subscript ranges are bound after declaration by looking at the number of elements and starting index, will be updated during run-time since it is dynamic.
4. Allocation of memory takes place in run-time in Python.
5. Python allows multidimensional arrays in array of arrays form.

```
-bash-4.2$ more my2.py
array = [0 for x in range(5)]

for x in range(0,5):
   array[x] = [0 for y in range(5)]
```

```
counter = 0
for x in range(0, len(array)):
   for y in range(0, len(array[0])):
      array[x][y] = counter
      counter += 1

print(array)

jagged = [["Gulsah"], ["Elif", "Esra"]]
print(jagged)
-bash-4.2$ python3 my2.py
[[0, 1, 2, 3, 4], [5, 6, 7, 8, 9], [10, 11, 12, 13, 14], [15, 16, 17, 18, 19], [20, 21, 22, 23, 24]]
[['Gulsah'], ['Elif', 'Esra']]
```

6. Python uses zero indexing. Maximum subscript: size-1
7. Pyhton does not have built-in support for arrays, it uses lists instead however there is ndarray from NumPy array stands for n- dimensional array. By using this library array objects can be initialized.
8. Slicing is supported in Python. It is in form name[first : end]

```
-bash-4.2$ more my3.py
array = [1, 2, 3, 4, 5, 6, 7, 8]
sliced = array[1:4]

print("array: ")
print(array)

print("now it is sliced: ")
print(sliced)
-bash-4.2$ python3 my3.py
array:
[1, 2, 3, 4, 5, 6, 7, 8]
now it is sliced:
[2, 3, 4]
```

Perl:

1. Integers are legal subscripts for Perl.

```
#!/usr/bin/perl

@nums = (1, 2, 3, 4, 5, 6);
@names = ("Elif", "Gulsah", "Kasdogan");

for($i = 0; $i < 6; $i = $i + 1){
   print "value of nums[$i]: $nums[$i]\n";
}
```

2. Perl has bound checking property
   #!/usr/bin/perl

   @nums = (1, 2, 3, 4, 5, 6);
   @names = ("Elif", "Gulsah", "Kasdogan");

   for($i = 0; $i < 6; $i = $i + 1){
      print "value of nums[$i]: $nums[$i]\n";
   }
   print $nums[80];

   print "done";
   Perl skips the element which is out of bounds and prints done. Bound check isbeing made and to avoid crashes or unintended results program ignores these. It does not give errors or warnings or it does not interrupt the program.
3. Subscript ranges are bound during run-time after declareation of array, since Perl array can grow by push and shrink by unshift ranges will be updated to aovid unintended memory accesses
4. Allocation takes place in run-time
5. Multidimensional arrays are in form array of arrays. Perl allows rectangular and jagged arrays.
   -bash-4.2$ more mypl3.pl
   #!/usr/bin/perl

   my @jagged = ([1, 2, 3, 4, 5, 6], [10,20,30], [5,6]);
   my @rectangular = ([1,2], [2,3]);

   print "jagged: \n";
   print "@$_\n" for @jagged;
   print "rectangular: \n";
   print "@$_\n" for @rectangular;
   -bash-4.2$ perl mypl3.pl
   jagged:
   1 2 3 4 5 6
   10 20 30
   5 6
   rectangular:
   1 2
   2 3
6. Arrays in Perl use zero indexing. Max subscript: size-1
7. Arrays are variables in Perl. We cannot initialize an array object.
8. Slicing is supported in Perl. It is in form @name[first..last]
   -bash-4.2$ more mypl4.pl
   #!/usr/bin/perl

```
@array = (1..10);

@sliced = @array[1..5];

print "original: @array\n";
print "sliced: @sliced\n";
-bash-4.2$ perl mypl4.pl
original: 1 2 3 4 5 6 7 8 9 10
sliced: 2 3 4 5 6
```

PHP:

1. User defined key which is type string and integers are legal subscripts for PHP arrays. PHP supports number indexed arrays, associative arrays and multidimensional arrays.

```
$numbers = array( 1, 2, 3, 4, 5,6,7,8);
foreach( $numbers as $i ) {
   echo "Value is $i <br />";
}

$letters[0] = "a";
$letters[1] = "b";

foreach( $letters as $i ) {
   echo "Value is $i <br />";
}

$ages = array("gulsah" => 20, "bilge" => 15, "ela" => 18);

echo "Gulsah's age: ". $ages['gulsah'] . "<br />";

$ages['gulsah'] = "twenty";

echo "Gulsah's age: ". $ages['gulsah'] . "<br />";

echo $numbers[90];

echo "done";
```

2. PHP uses run-time range checking.
   last line gives a warning because of the out of bounds. But it first prints done, displays the warning message afterwards. It does not interrupt the program.
3. It is bound at declaration, updated during run-time.
4. Allocation of array is done in run-time, arrays can grow and shrink in PHP by pop and push methods, allocation and deallocation will be made when necessary through execution.

5. Ragged and rectangular arrays are allowed in PHP

```
$rectangular = array(array(1,2,3), array(2,3,4), array(5,6,7));
echo $rectangular[0][1];

$jagged = array(array(1,2), array(8,9,10,11));
echo $jagged[1][1];
```

6. PHP uses zero indexing. Max subscript: size-1
7. Array can be created by usig array() in PHP. array item initialization can be in form :

```
$numbers = array( 1, 2, 3, 4, 5,6,7,8);
```

8. Slicing is supported in PHP. It is in form array_slice(name, first,len)

```
$nums = array(1,2,3,4,5,6,7,8,9);
echo "original:";
foreach( $nums as $i ) {
   echo " $i ";
}
$sliced = array_slice($nums, 1,4);
echo "sliced:";
foreach( $sliced as $i ) {
   echo " $i ";
}
```

Output: original: 1 2 3 4 5 6 7 8 9 sliced: 2 3 4 5