**CSF FINAL EXAM SOLUTIONS OF 00009115**

**Task 1**

<https://github.com/00009115/CSF.CW1.00009115/tree/exam>

**Task 2**

1. Dividing 9115 by 2 until the quotient is 0 to convert into *binary*:

|  |  |
| --- | --- |
| **Quotient**  9115 | **Remainder**  1 |
| 4557 | 1 |
| 2278 | 0 |
| 1139 | 1 |
| 569 | 1 |
| 284 | 0 |
| 142 | 0 |
| 71 | 1 |
| 35 | 1 |
| 17 | 1 |
| 8 | 0 |
| 4 | 0 |
| 2 | 0 |
| 1 | 1 |
| 0 |  |

911510 = 100011100110112

Dividing 9115 by 16 until the quotient is 0 to convert into *hex*:

|  |  |
| --- | --- |
| **Quotient**  9115 | **Remainder**  11 |
| 569 | 9 |
| 35 | 3 |
| 2 | 2 |
| 0 |  |

1110 = B16

911510 = 239B16

1. Dividing 9999 by 2 until the quotient is 0 to convert into *binary*:

|  |  |
| --- | --- |
| **Quotient**  99999 | **Remainder**  1 |
| 49999 | 1 |
| 24999 | 1 |
| 12499 | 1 |
| 6249 | 1 |
| 3124 | 0 |
| 1562 | 0 |
| 781 | 1 |
| 390 | 0 |
| 195 | 1 |
| 97 | 1 |
| 48 | 0 |
| 24 | 0 |
| 12 | 0 |
| 6 | 0 |
| 3 | 1 |
| 1 | 1 |
| 0 |  |

9999910 = 110000110100111112

911510 = 100011100110112

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| + | | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 |
|  |  |  | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 1 |
|  |  | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 0 |

11000011010011111 + 10001110011011 = 11010101000111010

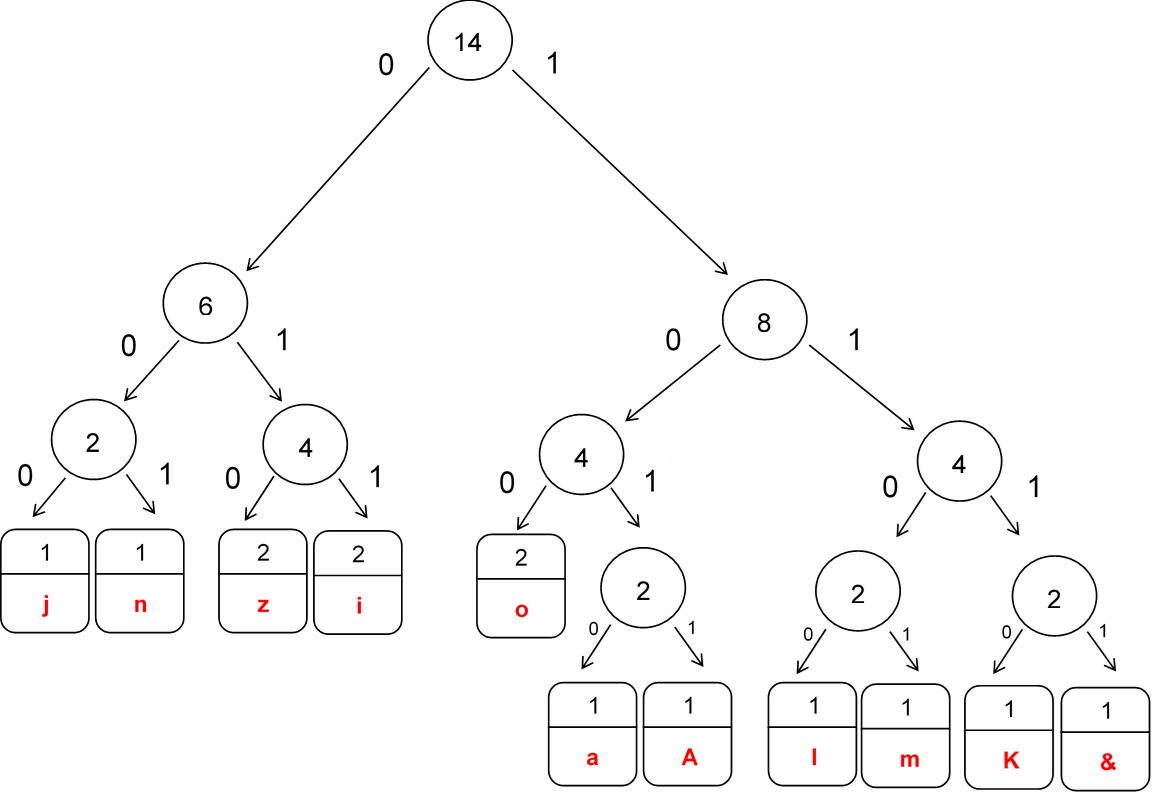
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| - | | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 |
|  |  |  | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 1 |
|  |  | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |

11000011010011111 - 10001110011011 = 10110001100000100

1. Hexadecimal or Hex numbers are widely used to denote colors in the so-called “hex” scheme. This is preferred over binary numbers because it is cleaner and can be defined with fewer digits. And the reason for choosing hex numbers over decimal numbers is that hex numbers have a base which is a multiple of 2 which can be used as a connection between hex and binary which decimal numbers do not have. This is important because hex numbers will be converted into the machine language – binary numbers at the end of the day.

**Task 3**

Aziza&Komiljon



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Character** | **Frequency** | **Code** | **Single Size (Bits)** | **Total size (Bits)** |
| z | 2 | 010 | 3 | 6 |
| i | 2 | 011 | 3 | 6 |
| o | 2 | 100 | 3 | 6 |
| j | 1 | 000 | 3 | 3 |
| n | 1 | 001 | 3 | 3 |
| a | 1 | 1010 | 4 | 4 |
| A | 1 | 1011 | 4 | 4 |
| l | 1 | 1100 | 4 | 4 |
| m | 1 | 1101 | 4 | 4 |
| K | 1 | 1110 | 4 | 4 |
| & | 1 | 1111 | 4 | 4 |
| TOTAL | 14 | - | 39 | 48 |

Encoded message: 101101001101010101111111010011010111100000100001

48 Bits in total

**Task 4**

Unordered: 9, 1, 1, 5, 4, 5, 2, 3, 7

Ordered: 1, 1, 2, 3, 4, 5, 5, 7, 9

The number to look for is: x=**2**

1) Midpoint is 4 (9/2=5.5, => 5th digit is the midpoint)

2) x<midpoint (2<4, ignore the range on the right: 1, 1, 2, 3, ~~4, 5, 5, 7, 9~~)

3) Midpoint is 1 (4/2=2 => 2nd digit in the remaining set is the midpoint)

4) x>midpoint (2>1, ignore the range on the left: ~~1, 1,~~ 2, 3)

5) Midpoint is 2 (2/2=1 => 1st digit in the remaining set is the midpoint)

6) x=midpoint (2==2) The number is found.

**Task 5**

Paged memory management uses fixed-size blocks which are called “pages” unlike other methods dividing the memory into various parts. In this methos, main memory is divided into same-sized blocks or frames where the data will be placed eventually. This method is used in order to achieve fast accessing processes and increasing the efficiency of the logical concept.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Page | 0 | 1 | 2 | 3 | 4 |
| Frame | 4 | 3 | 5 | 6 | 7 |

<2, 85> on the page of 2 we have **5** frames

We should calculate 5 frames of 1024-sized each + offset: 5 \* 1024 + 85 = 5205

1. <0, 1026> this logical address is invalid since it exceeds the frame size of 1024 (1024<1026)

**Task 6**

The software development methodoligies that are going to be differentiated below are Waterfall and Agile:

Waterfall methodology involves clear structure and plans for the future job processes. It has different solid and constant stages which cannot be come back to again. Several stages cannot overflow each other as well.

Agile methodology involves adaptavity and teamwork by allowing to come back to different stages and complete the particular tasks according to the conditions and needs. The overall process is divided into sprints and the feedback received at the end of each sprint is used during the next sprint. Scrum and Kanban, which are very popular techniques can be considered as Agile method of software development.

The main difference between Waterfall and Agile methodology is that Waterfall requires planning ahead while Agile values involvement and conditions.

**Task 7**

The network topologies that are going to be differentiated below are Bus Topology and Star Topology:

In Bus topology network is arranged using a single communication line for data transmission. This topology was used by early computer networking era representatives since it is basic and simple to implement.

Star topology is widely used in a modern life due to its simplicity and efficiency. In this topology, central node - hub takes place in the middle and all other dependent nodes are connected to each other through this hub.

The difference between them is that Bus topology is more sensitive for often network collapses on the main cable as it depends to single network cable while Star topolgy relies on a hub which is more easily controlled. It can be quite complicated to identify the collision reason in Bus topology since any part of the main cable can be damaged. However, Star topology does not allow other networks to break down because of certain cable damage and the issue can be addresed immediately by identifying the faied network part.

Advantages of Bus topology are easy installation and extension, less costly and less cables needed. Disadvantages of Bus topology are dependence on a single long cable network, difficulty on identifying the issue and all signals from all devices are transmissed at the same time through the same cable.

Advantages of Star topology are simplicity of troubleshooting and detecting the problem, single failed device does not affect the whole network, centralization allows efficient monitoring. Disadvantages of star topology are reliance on a hub, more cable need.

For home and small offices use, star topology is the very best option since it’s easy to implement and shows enough efficiency for the operations of these places. Bus topology is becoming less and less common due to the fact that it is the representative of primitive network topologies.

**Task 8**

# VOID FUNCTIONS  
# void functions do not return any value after being executed  
  
message = ''  
  
  
def check\_age\_void(age):  
 global message  
  
 if age < 18:  
 message = "You are " + str(age) + " years old, thus not eligible for the installment."  
 elif age >= 18:  
 message = "You are " + str(age) + " years old and eligible to receive an installment."  
 print(message)  
  
  
# Now the function above should be executed separately in order to update the message to be printed.  
if \_\_name\_\_ == '\_\_main\_\_':  
 check\_age\_void(16)  
  
# To avoid that, we can use value-returning functions which are shown below:  
  
  
# VALUE RETURNING FUNCTIONS  
  
def check\_age\_value(age):  
 if age < 18:  
 return "You are " + str(age) + " years old, thus not eligible for the installment."  
 elif age >= 18:  
 return "You are " + str(age) + " years old and eligible to receive an installment."  
  
# Now we can use the function as an argument of print() function as it returns string value  
  
  
print(check\_age\_value(20))

**Task 9**

# Functional  
def display\_operation\_func(name, balance, spent, gender): # function has 4 parameters  
  
 # Identifying the pronoun that will be used in the messages  
 if gender == "male":  
 pronoun = "his"  
 elif gender == "female":  
 pronoun = "her"  
 else:  
 pronoun = "their"  
  
 print("Client ", name, "spent an amount of", spent, "and", pronoun, "current balance is", balance-spent, "\n")  
  
  
display\_operation\_func("Yodgorov", 3750000, 250000, "female")

# calling the functions with arguments  
  
# OOP  
class User:  
 # creating a class which can be used to build new objects   
 def \_\_init\_\_(self, name, balance, spent, gender):  
 self.name = name  
 self.balance = balance  
 self.spent = spent  
 self.gender = gender  
  
 def identify\_pronoun(self):  
 if self.gender == "male":  
 return "His"  
 elif self.gender == "female":  
 return "Her"  
 else:  
 return "Their"  
  
 def update\_balance(self):  
 self.balance -= self.spent  
  
 def say\_operation(self):  
 print("Client ", self.name, "spent an amount of",  
 self.spent)  
  
 def say\_balance(self):  
 print(self.identify\_pronoun(), "current balance is", self.balance)  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
  
 # building the first object  
 user\_1 = User("Bakhodirov", 1500000, 84000, "male")  
 user\_1.update\_balance()  
 user\_1.say\_operation()  
 user\_1.say\_balance()  
  
 # building the second object  
 user\_2 = User("Azimova", 550000, 39000, "female")  
 user\_2.update\_balance()  
 user\_2.say\_operation()  
 user\_2.say\_balance()