TEAM MEMBERS :
ZYAD ATEF AL-ABIAD: 20231068
ALY ELDEEN YASEER ALI: 20231109
GAMAL MEGAHED SAYED: 20231039

Algorithm 1

```
# Check decimal validity
Function validate_decimal(binary)
for Loop through each character in the string
    If the character is not a valid decimal digit (0-9)
      Return False
  Return True

# Check binary validity
Function validate_binary(binary)
 forLoop through each character in the string
    If the character is not a valid binary digit (0 or 1)
      Return False
  Return True

# Check octal validity
Function validate_octal(octal)
  Convert the octal string to a list of characters
  forLoop through each character in the list
    If the character is not a valid octal digit (0-7)
      Return False
  Return True

# Check hexadecimal validity
Function validate_hexadecimal(hexadecimal):
  Convert the hexadecimal string to a list of characters
  forLoop through each character in the list
    If the character is not a valid hexadecimal digit (0-9, A-F)
      Return False
  Return True
```

Function to convert decimal to binary:

declare an empty list to store binary digits
While the decimal number is greater than:

Find the remainder when dividing the number by 2 and append it as a string to the list.
Divide the number by 2 and discard the remainder.
Reverse the list to get the correct binary representation.
Join the list elements into a string and return it as the binary equivalent.

Function to convert decimal to octal:

declare : i , res
i=o
res=0
While the decimal number is greater than 0:
Find the remainder when dividing the number by 8.
Add the remainder multiplied by 10 raised to the power of i to res.
Divide the number by 8 and discard the remainder.
Increment i.
Return the calculated res as the octal equivalent.

Function to convert decimal to hexadecimal:

Declare an empty list to store hexadecimal digits
While the decimal number is greater than 0:
Find the remainder when dividing the number by 16
If the remainder is less than 10
append it as a string to the list
else,
convert the remainder to its corresponding letter (A to F) using ASCII code and append it to the list
Divide the number by 16 and discard the remainder.
Reverse the list to get the correct hexadecimal representation.
Join the list elements into a string and return it as the hexadecimal equivalent.

Function to convert hexadecimal to decimal:

Declare : res , num
num = list(num)
num.reverse()
res = 0
for Loop through each character in the string
If the character is A to F
convert it to its corresponding decimal value using ASCII code.
else,
convert the character to its integer value.
Multiply the value by 16 raised to the power of the current index and add it to res.
Return the calculated res as the decimal equivalent.

Function to convert octal to decimal:

Declare : res , num , i
num =list (num)

res =0
i=o
While the octal number is greater than 0:
    Find the remainder when dividing the number by 10
    Add the remainder multiplied by 8 raised to the power of i to res
    Divide the number by 10 and discard the remainder
    Increment i.
Return the calculated res as the decimal equivalent.

Function to convert binary to decimal:

Declare : num , res
num=list (num)
num.reverse ()
res=0
for Loop through each character in the string
  Convert the character to its integer value (0 or 1).
  Multiply the value by 2 raised to the power of the current index and add it to res.
Return the calculated res as the decimal equivalent.

Menu (1)

While true :
Print a title for the numbering system converter
Print option A: "Insert a new number"
Print option B: "Exit program"

Get user's choice :
input choice (A OR B) from user to select an option

check validity of choice :
If the user entered "A"
  Prompt the user to enter a number.
    Start a validation loop:
      Check if the entered number is a valid hexadecimal number using a validation function (presumably defined else where)
    If the number is not valid:
      Prompt the user to enter a valid hexadecimal number.
        Repeat the validation loop.

If the user entered "B":
Exit the program.

Menu (2)

While true :
Declare : base

base = (A / B / C / D)
Print "Please select the base you want to convert a number from:"
Print "A) Decimal"
Print "B) Binary"
Print "C) Octal"
Print "D) Hexadecima

Get user’s choice :
Input (base) from the user

Check validity of choice :
If choice is "A" or "B" or "C" or "D":
    Break out of the loop
Else (choice is invalid):
     Display error message: "Please select a valid choice"
End of loop

Menu (3)

While true :
Print "Please select the base you want to convert a number to:"
Print "A) Decimal"
Print "B) Binary"
Print "C) Octal"
Print "D) Hexadecimal"

Get user's choice:
Prompt "(A/B/C/D)"
Read choice

Check validity of choice :
If choice is "A" or "B" or "C" or "D":
   Break out of the loop
Else (choice is invalid):
    Display error message: "Please select a valid choice"

End of loop

convert from decimal to decimal

```
      if base == A and base_convert == A
         while not validate_decimal(str(num))
            num = input(please enter a valid decimal number)
         print(The result:{num})
```

convert from decimal to binary

```
      elif base == A and base_convert == B
```

```
            while not validate_decimal(str(num))
                num = input(please enter a valid decimal number)
            print(The result:,dec_to_bin(num))
# convert from decimal to octal
        elif base == A and base_convert == C
            while not validate_decimal(str(num))
                num = input(please enter a valid decimal number)
            print(The result , dec_to_oct(num))
# convert from decimal to hexadecimal
        elif base == A and base_convert == D
            while not validate_decimal (str(num))
                num = input(please enter a valid decimal number)
            print(The result, dec_to_hexadecimal(num))
# convert from binary to decimal
        elif base == B and base_convert == A
# check validate binary
            while not validate_binary(str(num)):
                num = input(please enter a valid binary number)
            print(The result , bin_to_dec(num))
# convert from binary to binary
        elif base == B and base_convert == B
# check validate binary
            while not validate_binary(num)
                num = input(please enter a valid binary number)
            print(The result , num)
# convert from binary to octal
        elif base == B and base_convert == C
# check validate binary
            while not validate_binary(num):
                num = input(please enter a valid binary number)
# convert from binary to decimal then to octal
            print(The result , dec_to_oct(bin_to_dec(num)))
# convert from binary to hexadecimal
        elif base == B and base_convert == D
# check validate binary
            while not validate_binary(num):
                num = input(please enter a valid binary number)
# convert from binary to decimal then to hexadecimal
            print(The result , dec_to_hexadecimal(bin_to_dec(num))
# convert from octal to decimal
        elif base == C and base_convert == A
# check validate octal
            while not validate_octal(num):
                num = input(please enter a valid octal number)
```

```
        print(The result , octal_to_dec(num))
# convert from octal to binary
      elif base == C and base_convert == B
# check validate octal
        while not validate_octal(num):
           num = input(please enter a valid octal number)
# convert from octal to decimal then binary
        print(The result , dec_to_bin(octal_to_dec(num)))
# convert from octal to octal
      elif base == C and base_convert == C
# check validate octal
        while not validate_octal(num)
           num = input(please enter a valid octal number)
        print(The result , num)
```

Algorithm 2

```
# defining some function

# validity function
Declare : counter , check
Counter = 0
Check = len(binary)
Start a loop that continues as long as is_valid is False
    If the current character is not '0' or '1':
      Ask the user to enter a valid binary number
      Update the binary_string with the new input
     Break out of the inner loop.
   Else
       increase the counter by 1
If counter is equal to length:
  Set is_valid to True
   Return  binary

# check length
Measure the length of binary_one and binary_two
  If binary_one is longer
     Add leading zeros to binary_two until its length matches binary_one.
Else if binary_two is longer
     Add leading zeros to binary_one until its length matches binary_two.
Return: binary one , binary two
```

#two’s complement
Def two’s complement binary
Declare : check , counter two , two;s comple , counter , result
Store the length of the binary string in a variable called length
Create a copy of the binary string as a list of characters, called two_s_comple.
Set a counter variable counter_two to point to the last character in the list (index length - 1)
Start a loop that continues as long as counter_two is greater than or equal to 0:
If the character at two_s_comple[counter_two] is '1':
Decrement counter_two by 1
Break out of the loop
Else if it's '0':
Decrement counter_two by 1 and continue the loop
Start another loop that continues as long as counter_two is greater than or equal to 0:
If the current character is '0', change it to '1'
If the current character is '1', change it to '0'
Decrement counter_two by 1
Set a counter variable counter to 0
Create an empty string called result.
Start a loop that continues as long as counter is less than length:
Append the character at two_s_comple[counter] to the result string
Increment counter by 1
Return result

#start program- first menu
Print (binary calculator)
#body of program
Main loop :
Declare : option
Print A) Insert a number
Print B) Exit
ask the user to input their choice (A or B) and store it in a option
If option is "B":
Stop the program (break out of the loop).
while option is not "A" or "B":
Print(Please enter a valid option)
Print A) Insert a number
Print B) Exit
ask the user to input their choice (A or B) and store it in a option
declare : binary one
Ask the user to enter a binary number and store it in a binary_one.
Call the validate_binary function to check if binary_one is a valid binary number
If it's not valid, repeat steps until a valid binary number is entered

#second menu
Declare : choice

print(please select the operation)
 print(A) compute one's complement
 print(B) compute two's complement
 print(C) addition
 print(D) subtraction
ask the user to input their choice (A or B or C or D) and store it in a (choice)

#If input of secand menu is wrong

    while choice not in [A,B,C,D]
        print(Please enter a valid option)
        print(A) compute one's complement)
        print(B) compute two's complement)
        print(C) addition)
        print(D) subtraction)
        ask the user to input their choice (A or B or C or D) and store it in a (choice)

If the user chose option A:
  Declare a counter variable called counter_one and set it to 0.
  Declare a copy of the binary_one string as a list of characters, called one_s_comple.
  Declare : length
  Store the length of the binary_one string in a variable called length
Start a loop that continues as long as counter_one is less than length:
   If the current character in one_s_comple is '0', change it to '1'
   If the current character is '1', change it to '0'
        Increment counter_one by 1
Print (the one's complement =)
Start another loop that runs from 0 to length - 1:
    Print (one_s_comple[counter=]
    Increment counter by 1.
Print an empty line to move the cursor to the next line

Else If the user chose option B:
  Declare : result
  Call the two_s_complement function with binary_one as input and store the result in a (result)
  Print (the two s complement : , result)
Else if the user chose option C :
  Dclare :binary_two
  binary two= Input from the user (please enter the secand binary)

#summition
  Declare a counter variable counter = -1
  Declare an empty string called binary_sum to store the result
  Declare a variable called carry = 0 to hold the carry value

Start a loop that continues as long as counter is greater than or equal to -1 * the length of binary_one
Calculate the sum of the current bits and carry
Add the integer values of binary_one[counter], binary_two[counter], and carry
If bit_sum is 0:
Declare carry = 0
Declare : binary sum = 0 + binary sum
Else if bit_sum is 1:
carry = 0
binary sum = 1 + binary sum
Else if bit_sum is 2:
binary sum = 0 + binary sum
Set carry to 1
else
Append '1' to the beginning of binary_sum.
Set carry to 1
If carry is still '1' after the loop:
Append '1' to the beginning of binary_sum to account for the final carr
print (the summition of two binary num= binary_sum)

ELSE if choice is D :

```
else :
  binary_two = input (please enter the secand binary)

  #validity
  binary_two = validate_binary(binary_two)
  #length check
  binary_one , binary_two = length_check (binary_one, binary_two)
  #convert binary two to twos complement
  binary_two = two_s_complement(binary_two)
```

#differance#here we changed secend number to two s complement and add them

Declare : counter = -1 (to start from the rightmost digits)
Create an empty string called binary_diff to store the result
Declare : carry =0 to hold the borrow value (like a negative carry)
start a loop that continues as long as counter is greater than or equal to -1 * the length of binary_one
Calculate the difference of the current bits and borrow:
Add the integer values of binary_one[counter], the complement of binary_two[counter] (meaning 0 becomes 1 and 1 becomes 0), and carry
If bit_sum is 0:
Append '0' to the beginning of binary_diff
Set carry to '0'

```
    Else If bit_sum is 1:
      Append '1' to the beginning of binary_diff
      Set carry to '0'
   Else If bit_sum is 2:
       Append '0' to the beginning of binary_diff
       Set carry to '1' (indicating a borrow)
    Else :
         Append '1' to the beginning of binary_diff.
         Set carry to '1' (indicating a borrow)

print (the differance between the two binary num=binary_diff)
```