**(1)**

**Initial Solution:**

1. Initialize data as a string
2. Calculate the length of the data
3. Check the string is not empty
4. Initialize an array containing a number in words.
5. Check the length is equal to one
6. Iterate through the value.

**Refinement:**

1. Initialize the data as a string
2. Calculate the of the data
3. Check the string is not empty

3.1 if length equal to 0 then an empty string

1. Initialize an array

4.1 single digit array from zero to nine

4.2 two-digit array from ten to nineteen

4.3 tens multiple arrays from twenty and ninety

4.4 tens-power array hundred and thousand

1. If  the length is equal to one

5.1 print value from the single-digit array

1. Iterate through the value

6.1 if length greater than or equal to three

6.1.1 if num[x]-'0' is not equal to zero

                            6.1.1.1  print single\_digits[num[x] - '0']

                                    6.1.1.2 print tens\_power[len - 3]

                         6.1.2 decrement the length

            6.2 else

6.2.1 if num[x]-'0' is equal to one

6.2.1.1 assign sum is equal to num[x]-'0' + num[x]-'0'

6.2.1.2 print two\_digits[sum]

                        6.2.2 else if num[x]-'0' is equal to 2 and num[x+1]-'0' is equal to 0

6.2.2.1 print twenty

6.2.3 else

6.2.3.1 assign i=num[x]-'0'

6.2.3.2 if i greater than zero

6.2.3.2.1 print tens\_multiple[i]

6.2.3.3 increment x by one

6.2.3.4 if num[x]-'0' not equal to zero

6.2.3.4.1 print single\_digits[num[x]-'0']

6.3 increment x by 1

**Procedural Abstraction:**

procedure convert\_to\_words(num)

len =num.length

if(len==0)

print("empty string");

single\_digits[]={ "zero", "one", "two", "three", "four", "five", "six", "seven","eight", "nine"};

   two\_digits={"", "ten", "eleven", "twelve","thirteen", "fourteen",

"fifteen", "sixteen","seventeen","eighteen", "nineteen"};

tens\_multiple={"", "", "twenty","thirty","forty","fifty","sixty",     "seventy","eighty", "ninety"};

tens\_power={"hundred", "thousand"};

print (num);

if(len=1)

print(num[0]-'0');

while(x < num.length)

            if(len>=3)

                   if (num[x]-'0' != 0)

            print(single\_digits[num[x]-'0'])

print(tens\_power[len-3])

                           --len;

            if(num[x]-'0=1)

           int sum = num[x] - '0' + num[x] - '0';

print(two\_digits[sum]);

           elseif(num[x]-'0'==2&&num[x + 1] - '0' == 0)

            print("twenty");

           else

                i=(num[x]-'0');

                if(i > 0)

                print(tens\_multiple[i]);

                else

                print("");

                ++x;

                if(num[x]-'0'!= 0)

                    print(single\_digits[num[x] - '0']);

            ++x;

   End procedure

**(2)**

**Initial Solution:**

1. Initialize the value
2. Assign tolerance=1e-8 and max\_tier=50
3. Assign i=0
4. Derive the equation

**Refiniment:**

1. Initialize the value
2. Assign tolerance=1e-8 and max\_tier=50
3. Assign i=0
4. Iterate till fabs(x-x\_old)/x greater than tolerance and i less than max\_tier

4.1 assign x\_old=x

4.2 assign x=2-0.5\*sin(x)

4.3 increment i by 1

**Procedural Abstraction:**

procedure equation()

tolerance=le-8

max\_iter=50

x=1.0

i=0

do

x\_old = x

x = 2 - 0.5\*sin(x)

while(fabs(x-x\_old)/x > TOLERANCE && i < MAX\_ITER)

end procedure

**(3)**

**Initial Solution:**

1. Initialize process along with burst time
2. Find waiting time
3. Find turnaround time
4. Find average waiting time
5. Find average turn around time

**Refinement:**

1. Initialize process along with burst time
2. Find waiting time

2.1 If first process

2.1.2 assign wt[0] as 0

2.2 iterate

2.2.1 wt[i]=bt[i-1]+wt[i-1]

1. Find turnaround time

3.1 turnaround time = waiting\_time+burst\_time

1. Find Average waiting time

4.1 Average waiting time=total\_waiting\_time/no\_of\_processes

1. Find average turnaround time

5.1 average turnaround time=total\_turn\_around\_time/no\_of\_processes.

**Procedural Abstraction:**

procedural findwaitingtime(processes, n, bt, wt)

wt[0]=0

for  i<n

wt[i]=bt[i-1]+wt[i-1];

end procedure

procedure findturnaroundtime(processes, n, bt, wt, tat)

for i<n

tat[i]=bt[i]+wt[i];

end procedure

procedure findavgtime(processes, n, bt)

findwaitingtime(processes,n,bt,wt)

findturnaroundtime(processes,n,bt,wt,tat)

print processes, burst time, waiting time, turnaround time

for i<n

total\_wt=total\_et+wt[i]

total\_tat = total\_tat+tat[i];

print bt[i],wt[i],tat[i]

print total\_wt/n

print  total\_tat/n

end procedure

**Step-by-step explanation**

The stepwise refinement approach is nothing but the steps to implement the complex program into a subprogram. In the answer part, the initial solution explains what to do in the program basically but in the refinement part, explains each every step that was very useful to developers to develop the program without missing any module.

Procedural abstraction is used to describe an instruction clearly as much as possible.

**Reference:**

GeeksforGeeks. 2020. *Program To Convert A Given Number To Words - Geeksforgeeks*. [online] Available at: <https://www.geeksforgeeks.org/convert-number-to-words/> [Accessed 8 October 2020].