*Logic Specification Template*

|  |  |  |  |
| --- | --- | --- | --- |
| **Student** | David Zavala López | **Program #** | 5 |

**Include the image of the activity diagram created for the program:**

Clase: Simpsons

|  |  |
| --- | --- |
| **Nombre** | **Parámetros** |
| Gamma | x: float >= .5 |
| If x = 1 -> return = 1  If x = 0.5 -> return = sqrt(pi)  return = (x-1) \* Gamma(x-1) | |
| TDistributionPDF | x: int >= 0  dof: int > 0 |
| numerator := Gamma((dof + 1) / 2)  denominator := (Gamma(dof / 2) \* ((dof \* pi) \*\* 0.5)))  return = (nominator / denominator) \* (1 + (x \*\* 2 / dof)) \*\* (-(dof + 1) / 2) | |
| SimpsonsRuleIntegration | func: método que represente una función aplicable para el caso  n: int > 1  W: float -> x/n  x : float >= 0 |
| sumOdds := GetSumOdds()  sumEvens := GetSumEvens()  integral := (W / 3) \* (func(0,dof) + sumOdds + sumEvens + func(x,dof))  return = integral | |
| GetSumOdds(func, n, W, dof) | func: método que represente una función aplicable para el caso  n: int > 1  W: float -> x/n  dof: int > 0 |
| sumOdds := 0  ∀ i ∈ oddNumbers : 0 .. n – 1 | sumOdds := sumOdds + (4 \* simpson(i \* W, dof))  return = sumOdds | |
| GetSumEvens(func, n, W, dof) | func: método que represente una función aplicable para el caso  n: int > 1  W: float -> x/n  dof: int > 0 |
| sumEvens := 0  ∀ i ∈ evenNumbers : 0 .. n – 1 | sumEvens := sumEvens + (2 \* simpson(i \* W, dof))  return = sumEvens | |
| IntegrateWithSimpsonsRule | x: float >= 0  dof: int > 0  n: int > 1 |
| E := .0000001  probability1 := SimpsonsRuleIntegration(TDistributionPDF, n, x/n, x, dof)  n := n\*2  probability2 := SimpsonsRuleIntegration(TDistributionPDF, n, x/n, x, dof)  if abs (probability2 – probability1) > E ->  probability1 = probability2;  n \* 2;  repeat;  return = probability2 | |
| FindXBySimpsonsRule | p:float >= 0, <= 0.5  dof:int > 0  n:int > 1  x:float >= 0  d:float > 0  E:float > 0  lastResult:float >= 0  posOfLastP:str >= 0 |
| pFromCalculation := SimpsonsRuleIntegration(TDistributionPDF, n, x/n, x, dof)  if abs (probability2 – probability1) > E ->  if pFromCalculation < p ->  if posOfLastP != “Lower” && posOfLastP != “” ->  d := d/2  x := x+d  return FindXBySimspsonsRule(p,dof,n,x,d,E,pFromCalculation,"Lower")  if pFromCalculation > p ->  if posOfLastP != “Higher” && posOfLastP != “” ->  d := d/2  x := x+d  return FindXBySimspsonsRule(p,dof,n,x,d,E,pFromCalculation,"Higher")  return x | |
| CalculateX | x: float >= 0  dof: int > 0  n: int > 1 |
| if fileExists(“Cache/”+str(x)+’-‘+str(dof)+’.txt’) ->  open(“Cache/”+str(x)+’-‘+str(dof)+’.txt’)  return = file  else ->  p := IntegrateWithSimpsonsRule(x, dof, n)  Imprimir resultados en el formato indicado  saveResultInCache(x,dof,p) | |

Clase: Main

|  |  |
| --- | --- |
| **Declaración** | **Parámetros** |
| main() |  |
| p := askForFloatInput("Ingrese valor de p: ",0,True,0.5,True)  dof := askForIntInput("Ingrese valor de dof (grados de libertad): ",0)  CalculateX(p,dof) | |

Clase: StringHelper

|  |  |
| --- | --- |
| **Declaración** | **Parámetros** |
| isNumber(string) | string: string |
| acceptedChars = ['0','1','2','3','4','5','6','7','8','9',',','.']  Iterar char sobre string:  Si char no esta en acceptedChars:  return = False  return = True | |
| removeCommas(string) | string: string |
| newValue = “”  Iterar char sobre string  Si char ¡= ‘,’:  Agregar char a newValue  return = newValue | |
| convertToFloat(string) | string: string |
| Intentar:  f := float(string)  return = f  Si no se pudo:  Avisar del error en pantalla  Regresar un valor nulo | |
| convertToInt(string) | string: string |
| Intentar:  f := int(string)  return = f  Si no se pudo:  Avisar del error en pantalla  Regresar un valor nulo | |
| isBlank(string) | string: string |
| Si la linea no esta vacía después de limpiar espacios, tabuladores y enters  regresar false  regresar true | |
| extractCant(string) | string: string |
| string := string.strip()  string := string.strip  return := string | |

Clase: InputHelper

|  |  |
| --- | --- |
| **Declaración** | **Parámetros** |
| askForFloatInput | message:str  min:float  minAccepted:bool  max:float  maxAccepted:bool |
| Iteración infinita:  Si el usuario mando un message  Imprimir ese mensaje y pedir input  Si no hay un mensaje mandado  Imprimir un mensaje por defecto  inputValue = input del usuario  Si inputValue está en blanco:  Volver a empezar la iteración  Si inputValue no es un número:  Volver a empezar la iteración  Remover comas de inputValue  inputAsNumber := Convertir inputValue en un float  Si inputAsNumber es nulo:  Volver a empezar la iteración  Si inputAsNumber es menor que min:  Volver a empezar la iteración  Si inputAsNumber es mayor que max:  Volver a empezar la iteración    return = inputAsNumber | |
| askForIntInput | message:str  min:float  minAccepted:bool  max:float  maxAccepted:bool |
| Iteración infinita:  Si el usuario mando un message  Imprimir ese mensaje y pedir input  Si no hay un mensaje mandado  Imprimir un mensaje por defecto  inputValue = input del usuario  Si inputValue está en blanco:  Volver a empezar la iteración  Si inputValue no es un número:  Volver a empezar la iteración  Remover comas de inputValue  inputAsNumber := Convertir inputValue en un int  Si inputAsNumber es nulo:  Volver a empezar la iteración  Si inputAsNumber es menor que min:  Volver a empezar la iteración  Si inputAsNumber es mayor que max:  Volver a empezar la iteración    return = inputAsNumber | |