# TND - Partie de Jessy

## Partie chef de projet et planification

* Jordan :
  + Fonction principale
    - A finir pour le 30/04
  + Prototype des fonctions
    - A finir pour le 25/04
* Dany :
  + Fonction « CalculNombreItération »
    - A finir pour le 25/04
  + Fonction « CalculErreurRelativeMax »
    - A finir pour le 25/04
  + Fonction « CalculNombreDécimalesNécessairePourChaqueTerme »
    - A finir pour le 25/04
  + Fonction « Calcul »
    - A finir pour le 30/04
  + Fonction « MultipleDe05 »
    - A finir pour le 25/04
  + Fonction « DegréN »
    - A finir pour le 25/04

## Partie analyse

L’objectif pour cette partie était d’implémenter en langage C le calcul de , et cela dans n’importe quelle base entière comprise entre 1 et 10.

Il a d’abord fallu réfléchir à quelle méthode utilisé pour réaliser ce calcul : ou bien adapter le calcul principal de pour le faire fonctionner selon n’importe quelle base, ou bien transformer dès le début l’exposant pour pouvoir utiliser la formule permettant de calculer . Ici, j’ai pris la décision de transformer dés le début le calcul pour ne plus avoir à gérer la base.

Ensuite, il a fallu réfléchir à toutes les possibilités d’utilisation de ce programme : on peut rentrer des exposants entiers ou à virgules, négatif ou positif. Il a donc fallu réfléchir à comment gérer les différents cas d’utilisations.

Pour gérer les nombres négatifs, rien de plus simple : il suffit dés lors de retenir via un Boolean que l’exposant était négatif, réaliser les calculs sur l’opposé de l’exposant, et a la fin diviser 1 par le résultat obtenu.

Pour gérer les nombres positifs, qu’ils soient entiers ou à virgule, un problème se posait : le calcul d’exponentielle n’est suffisamment fiable dans ses résultats que dans l’intervalle compris entre 0 et 0.5. Mais grâce aux propriétés de l’exponentielle, pour un nombre supérieur à 0.5 il suffit de calculer , de multiplier ce résultat par lui-même autant de fois qu’il y avait 0.5 dans l’exposant de base, puis de multiplier ce résultat par l’exponentielle du reste en base e.

Un dernier problème était alors à gérer : la propagation des erreurs. En effet, chaque calcul d’exponentielle à sa précision, et multiplier des exponentielles entre elles réduisent ladite précision. Il suffit alors de calculer le nombre de décimales nécessaire pour chaque terme pour s’assurer d’une précision suffisante par rapport a la demande, et de calculer le nombre d’itérations nécessaire pour obtenir suffisamment de bonnes décimales pour chaque terme.

## Partie code

┌─── \*Calcul de a^x

│ Obtenir base, exposant, nbDecimales

│ exposant \*= ln(base)

│┌── if (exposant < 0)

││ negatif = true

││ exposant \*= -1

│└──

│┌── if (exposant < 0.5)

││ o───────────────────────o ↓ nbDecimales,exposant

││ │ CalculNombreItération │

││ o───────────────────────o ↓ nbItérations

││ o────────o ↓ exposant,nbItérations

││ │ Calcul │

││ o────────o ↓ résultat

│├── else

││ o─────────────────o ↓ exposant

││ │ EstMultipleDe05 │

││ o─────────────────o ↓ multipleDe05

││┌── if (multipleDe05)

│││ argMod = exposant

││├── else

│││ argMod = (((exposant \* 2) + 1)ENT ) / 2

││└──

││ o────────────────────────────────────────────────o ↓ argMod,nbDécimales

││ │ CalculNombreDécimalesNécessairePourChaqueTerme │

││ o────────────────────────────────────────────────o ↓ nbDécimalesNécessaires

││ o──────────────────────────o ↓ nbDécimalesNécessaires,exposant

││ │ ObtentionNombreItération │

││ o──────────────────────────o ↓ nbItérations

││ nombreDeFois05 = (exposant \*2)ENT

││ reste = exposant - (0.5 \* nombreDeFois05)

││ arg = 0.5

││ o────────o ↓ arg,nbItérations

││ │ Calcul │

││ o────────o ↓ résultatPartiel

││ résultat = 1

││ i = nombreDeFois05

││╔══ do while (i > 0)

││║ résultat \*= résultatPartiel

││║ i--

││╙──

││┌── if (reste > 0)

│││ arg = reste

│││ o────────o ↓ arg,nbItérations

│││ │ Calcul │

│││ o────────o ↓ résultatDuReste

│││ résultat \*= résultatDuReste

││└──

│└──

│┌── if (negatif)

││ résultat = 1 / résultat

│└──

│ Sortir résultat

└──────────

┌─── \*CalculNombreItération

│ o─────────────────────────o ↓ nbDécimales

│ │ CalculErreurRelativeMax │

│ o─────────────────────────o ↓ erreurRelativeMax

│ nbItérations = 1

│ essai = ( (exposant / 2) \* (0.5 ^ (nbItérations + 1) ) ) / ( (nbItérations + 1) ! )

│╔══ do while (essai > erreurRelativeMax)

│║ nbItérations++

│║ essai = ( (exposant / 2) \* (0.5 ^ (nbItérations + 1) ) ) / ( (nbItérations + 1) ! )

│╙──

│ Sortir nbItérations

└──────────

┌─── \*CalculErreurRelativeMax

│ erreurRelativeMax = 0.5

│ nombre = 10

│╔══ do while (nbDecimalesNécessaire > 0)

│║ nombre \*= 10

│║ nbDecimalesNécessaire--

│╙──

│ Sortir erreurRelativeMax \*= (1 / nombre)

└──────────

┌─── \*CalculNombreDécimalesNécessairePourChaqueTerme

│ p = (( nbDécimales + (argMod - 0.5) \* 0.5 + (argMod/5) + 0.31 ) + 1)ENT

│ Sortir p

└──────────

┌─── \*Calcul

│ résultat = 1

│ degré = 1

│╔══ do while (degré ≤ nbItérations)

│║ o────────o ↓ arg, degré

│║ │ DegréN │

│║ o────────o ↓ résultatDegréI

│║ résultat += résultatDegréI

│║ degré++

│╙──

│ Sortir résultat

└──────────

┌─── \*MultipleDe05

│╔══ do while (exposant > 0)

│║ exposant -= 0.5

│╙──

│┌── if (exposant == 0)

││ multipleDe05 = true

│├── else

││ multipleDe05 = false

│└──

│ Sortir mutlipleDe05

└──────────

┌─── \*DegréN

│ Sortir (arg ^ degré) / (degré !)

└──────────

## Partie testing

Titre : Fonction testée

Libellé Argument nombre de résultat résultat conclusion

Du test Encodé décimales attendu obtenu (ok ou non)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Titre : Calcul de a^x | | | | |
| Libéllé du test | Argument encodé | Nombre de décimales | Résultat attendu | Résultat conclusion |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |