Exercice 1

Problème

Écrire l'algorithme d'un sous-programme qui calcule la valeur approchée de l'intégrale de f(x) = 1/x entre deux bornes strictement positives a et b (b > a > 0).

Documentation

I : réel, l'intégrale de 1/x entre a et b avec b > a > 0 approchée par la méthode des trapèzes, c'est une aire géométrique > 0
a : un réel, borne inférieur de l'intégrale a > 0
b : un réel, borne supérieur de l'intégrale b> a
n : un entier >0, le nombre de trapèze utilisée pour approcher l'intégrale, il correspond à la précision de l'approximation.
h :un réel, la hauteur d'un trapèze
x, x' : deux réels, les bornes du i^{ième} trapèze
U : un réel, l'aire du i^{ième} trapèze

Algorithme

Résultat : I
Données : a,b,n

Etape 1 :

$I = \sum_{i=0}^{n-1} A_i$ avec Ai l'aire du i^{ème} trapèze et sachant que l'aire d'un trapèze est (b+B)*h/2

Hauteur du trapèze h = (b-a) /n

$A_i = f(x_i) + f(x_{i-1}) / 2 * h = (1/x_i + 1/x_{i-1}) * h /2$

$x_{i+1} = x_i + h$

$$I = \sum_{i=0}^{n-1} \frac{\frac{1}{x_i} + \frac{1}{x_{i+1}}}{2} * h \quad \text{avec } x_{i+1} = x_i + h$$

$$I = \sum_{i=0}^{n-1} \frac{\frac{1}{x_i} + \frac{1}{x_i + h}}{2} * h$$

Étape 2 :

La récurrence pour la somme :

Pour i de 0 à n-1

$$\begin{cases} I_0 = 0 \\ I_i = I_{i-1} + A_i \text{ avec } A_i \text{ aire du i}^{ième} \text{ trapèze} \end{cases}$$

Étape 3 :

h ← (b-a) / n
x ← a
I ← 0
Pour i de 0 à n-1 faire

x’ ← x + h
U ← (1/x + 1/x’) * h / 2
I ← I + U
x ← x‘

Faire

Exercice 2

Problème

Écrire l'algorithme d'un sous-programme « *stringCmp* » qui compare deux chaînes de caractère selon l'ordre lexicographique (relation d'ordre total).

Lexique – Documentation

R : *Enuméré (1 :S1<S2, 2 : S1=S2, 3: S1>S2)*
s1, s2 : deux chaînes de caractères, tableau de caractères indexées au 0 à *sizeS1|sizeS2 -1*
sizeS1, sizeS2 : deux entiers strictement positifs
i : un entier, variable d'itération strictement positif

Algorithme

Résultat : R
Données : s1, s2, *sizeS1, sizeS2*

Fonction stringCmp (s1 : chaîne indexée de 0 à sizeS1 -1, sizeS1 : Entier, S2 : chaine indexée de 0 à sizeS2 -1, sizeS2 : Entier) : Enuméré (1 :S1<S2, 2 : S1=S2,3: S1>S2)

```
Début
    i ← 0
    Tant que (i<sizeS1) ET (i< sizeS2) ET (s1[i] = s2[i]) faire
        i ← i+1
    Fait

    Si (i=sizeS1) ET  i<sizeS2) alors
        stringCmp ← 1
    sinon
        Si (i=sizeS2) ET  i<sizeS1) alors
        stringCmp ← 3
        sinon
            Si (i=sizeS2) ET  (i=sizeS1) ET (s1[sizeS1-1] = s2[sizeS2-1]) alors
                stringCmp ← 2
            sinon
                Si (s1[i] < s2[i]) alors
                    stringCmp ← 1
                alors
                    stringCmp ←3
                FinSi
            FinSi
        FinSi
    FinSi
Fin.
```

Code C

```
int stringCmpASCII(int sizeS1, char* s1, int sizeS2, char* s2) {
	int i = 0;

	while ( (i<sizeS1) && (i<sizeS2) && (s1[i] == s2[i]) ) {
		i++;
	}

	if ((i==sizeS1) && (i < sizeS2)) {
		return -1;
	} else {
		if ((i==sizeS2) && (i < sizeS1)) {
			return 1;
		} else {
			if ((i==sizeS2) && (i == sizeS1) && (s1[i-1] == s2[i-1]) ) {
				return 0;
			} else {
				if(s1[i] < s2[i]) {
					return -1;
				} else {
					return 1;
				}
			}
		}
	}
}


int stringCmpASCIIWithoutLength(char* s1, char* s2) {
	int i = 0;

	while ( (s1[i]!='\0') && (s2[i]!='\0') && (s1[i] == s2[i]) ) {
		i++;
	}

	if ((s1[i]=='\0') && (s2[i]!='\0')) {
		return -1;
	} else {
		if ((s1[i]!='\0') && (s2[i]=='\0')) {
			return 1;
		} else {
			if ((s1[i]=='\0') && (s2[i]=='\0') && (s1[i-1] == s2[i-1])) {
				return 0;
			} else {
				if(s1[i] < s2[i]) {
					return -1;
				} else {
					return 1;
				}
```

```
                    }
                }
            }
}
```

Voir reste code C pour les versions génériques avec relation d’ordre total sur les caractères en paramètre.

Exercice 3

Problème

Écrire l'algorithme du sous-programme « flag » qui permet de trier une séquence de N couleurs {Bleu B, Blanc W, Rouge R}. La séquence triée regroupe d'abord tous les rouges, puis tous les blanc puis tous les bleus.

En entrée : W R B B R R W W R
En sortie : B B W W W R R R R
Contrainte : utiliser qu'une seule boucle

Lexique – Documentation

T : un tableau de couleur {B,W,R} indexée de 0 à N-1
N : un entier strictement positif
b : un entier, l'index du dernier bleu dans T strictement positif
r : un entier, l'index du premier rouge dans T strictement positif
i : un entier, variable d'itération strictement positif

Algorithme

Résultat : T trié
Données : T non trié, N

```
Fonction flag (N : un entier >0, T : un tableau de couleur non trié) : un tableau de couleur trié
Début
b ← 0
r ← N-1
i ← 0
Tant que (i<=r) faire
        Si T[i] = bleu alors
                T[i] ← T[b]
                T[b] ← bleu
                b ← b +1
                i ← i +1
        sinon
                Si T[i] = rouge alors
                        T[i] ← T[r]
                        T[r] ← rouge
                        r ← r - 1
                sinon
                        i ← i+1
                finSi
        finSi
fait
flag ← T
Fin.
```

Exercice 4

Problème
Écrire l'algorithme d'un sous-programme qui remplace dans une chaîne de N caractères tous les multiples espaces consécutSis par un espace unique.

En entrée: - - - Zer - - - erzerez - r - -
En sortie: - Zer - erzerez - r -

Lexique – Documentation
S : une chaîne de caractères terminant par '\0'
i,j : deux entiers >0

Algorithme
Résultats : S avec espaces multiples
Données : S sans espaces multiples

Fonction removeDirtySpaces(stringWithSpaces : String) : String (modified version of the input string)
Début
    i : Entier ← 0
    j : Entier ← 0
    hasRemoved : Boolean ← FALSE // utile uniquement si on veut gérer le cas d'une entrée sans aucun espace
    Tantque (stringWithSpaces[i] <> '\0') faire
        hasRemoved ← FALSE
        Tantque (stringWithSpaces[i] <> '\0') AND (stringWithSpaces[i] <> ' ') faire
            //iterating on non-space characters
            stringWithSpaces[j] ← stringWithSpaces[i]
            i ← i+1
            j ← j+1
        fait

        Tantque (stringWithSpaces[i] <> '\0') AND (stringWithSpaces[i] = ' ') faire
            //iterating on consecutive spaces
            hasRemoved ← TRUE
            i ← i+1
        fait
        Si (hasRemoved) alors
            stringWithSpaces[j] ← ' '
            j ← j+1
        finSi
    fait
    stringWithSpaces ← realloc(stringWithSpaces, j)
    removeDirtySpaces <- stringWithSpaces
Fin.

Code C
```
void removeDirtySpaces(char* stringWithSpaces) {
```

```
	int i = 0;
	int j = 0;
	Boolean hasRemoved = FALSE;
	while (stringWithSpaces[i] != '\0') {
		hasRemoved = FALSE;

		while ((stringWithSpaces[i] != '\0') && (stringWithSpaces[i] != ' ')) {
			stringWithSpaces[j]=stringWithSpaces[i];
			i++;
			j++;
		}
		while ((stringWithSpaces[i] != '\0') && (stringWithSpaces[i] == ' ')) {
			i++;
			hasRemoved = TRUE;
		}
		if (hasRemoved) {
			stringWithSpaces[j] = ' ';
			j++;
		}
	}

	/* Erasing the end of the String */
	stringWithSpaces = (char*) realloc(stringWithSpaces,j*sizeof(char));
	stringWithSpaces[j] = '\0';
}
```