Structure de données TP4

Table des matières

[Exercice 1 2](#_Toc89185992)

[Algorithme 2](#_Toc89185993)

[Code C 2](#_Toc89185994)

[Exercice 2 3](#_Toc89185995)

[Algorithme 3](#_Toc89185996)

[Code C 3](#_Toc89185997)

[Exercice 3 3](#_Toc89185998)

[Algorithme 3](#_Toc89185999)

[Code C 4](#_Toc89186000)

[Exercice 4 4](#_Toc89186001)

[Algorithme 4](#_Toc89186002)

[Code C 6](#_Toc89186003)

## Exercice 1

### Algorithme

add(a : entier, b : entier) : entier

Début

    retourner a+b

Fin

multiply(a : entier, b : entier)

Début

    Si (a==0 || b==0)

       retourner 0

    Sinon si (a == 1)

       retourner b

    Sinon si (b == 1 )

       retourner a

    Sinon

        retourner a + multiply (a, b-1)

    Fin\_si

Fin

### Code C

#include <stdio.h>

#include <stdlib.h>

int add (int a, int b){

    return a+b ;

}

int multiply(int a, int b){

    if (a==0 || b==0)

        return 0;

    else if (a == 1)

        return b ;

    else if (b == 1 )

        return a ;

    else{

        return  a + multiply (a, b-1) ;

    }

}

## Exercice 2

### Algorithme

display (n : entier) : entier

Début

   si (n == 1)

        printf("%d \n", n)

        retourner 1

    Sinon

        printf("%d \n", 1+display(n-1))

        retourner n

    fin\_si

Fin

### Code C

#include <stdio.h>

int display (int n){

    if (n == 1) {

        printf("%d \n", n) ;

        return 1 ;

    }

    else{

        printf("%d \n", 1+display(n-1))  ;

        return n;

    }

}

int main(int argc, char const \*argv[])

{

    display(30) ;

    return 0;

}

## Exercice 3

### Algorithme

Ici, le code parle de lui-même.

Une méthode récursive facile en trois lignes assez connu.

### Code C

//Solution récursive :

void hanoi(int n, char piquetDebut, char piquetFin, char piquetLibre) {

    if (n > 0) {

        hanoi(n-1,piquetDebut,piquetLibre,piquetFin);

        printf("Tour %c a tour %c\n",piquetDebut,piquetFin);

        hanoi(n-1,piquetLibre,piquetFin,piquetLibre);

    }

}

int main(int argc, char \*argv[])

{

    if (argc == 2) {

        int n = atoi(argv[1]);

        hanoi(n,'A','C','B');

    }

    return 0;

}

## Exercice 4

### Algorithme

fonction factor(str : chaîne, length : entier, position : entier) : booléen

début

    booléen valide = vrai

    si str[position] != 'a' alors

        si str[position] != '(' alors

            valide = faux

        sinon

            position ++

            si expr(str,length,position) == vrai alors

                si str[position] == ')' alors

                    position ++

                sinon

                    valide = faux

                fin\_si

            sinon

                valide = faux

            fin\_si

        fin\_si

    sinon

        position ++

    fin\_si

    retourner valide

fin

fonction term(str : chaîne, length : entier, position : entier) : booléen

début

    booléen valide = vrai

    si factor(str,length,position) == vrai alors

        si str[position] == '\*' alors

            position ++

            si factor(str,length,position) == faux alors

                valide = faux

            fin\_si

        fin\_si

    sinon

        valide = faux

    fin\_si

    retourner valide

fin

fonction expr(str : chaîne, length : entier, position : entier) : booléen

début

    booléen valide = vrai

    si term(str,length,position) == vrai alors

        si str[position] == '+' alors

            position ++

            si term(str,length,position) == faux alors

                valide = faux

            fin\_si

        fin\_si

    sinon

        valide = faux

    fin\_si

    retourner valide

fin

fonction test(str : chaîne, length : entier, position : entier) : booléen

début

    booléen valide = faux

    si expr(str,length,position) == vrai ET legnth == position alors

        valide = vrai

    fin\_si

    retourner valide

fin

### Code C

#include "string.h"

#include <stdio.h>

int expr(char \* str, int length, int\* pos);

int factor(char \*str, int length, int\* pos) {

    int valid = 1;

    if (str[\*pos] != 'a') {

        if (str[\*pos] != '(') {

            valid = 0; // pas de parenthese ni de a, pas bon

        } else {

            // parenthese trouvé, on check une expression

            \*pos = \*pos + 1;

            if (expr(str,length,pos) == 1) {

                if (str[\*pos] == ')') { // deuxieme parenthèse

                    \*pos = \*pos + 1;

                } else valid = 0;

            } else valid = 0;

        }

    } else { // si c'est un a, on passe au symbole suivant

        \*pos = \*pos + 1;

    }

    return valid;

}

int term(char \*str, int length, int\* pos) {

    int valid = 1;

    if (factor(str,length,pos) == 1) {

        if(str[\*pos] == '\*') {

            \*pos = \*pos + 1;

            if (factor(str,length,pos) != 1) {

                valid = 0;

            }

        }

    } else {

        valid = 0;

    }

    return valid;

}

int expr(char \* str, int length, int\* pos) {

    int valid = 1;

    if (term(str,length,pos) == 1) {

        if(str[\*pos] == '+') {

            \*pos = \*pos + 1;

            if (term(str,length,pos) != 1) {

                valid = 0;

            }

        }

    } else {

        valid = 0;

    }

    return valid;

}

int test(char\* str, int length, int\* pos ) {

    int valid = 0;

    if (expr(str,length,pos) == 1 && length == \*pos) {

        valid = 1;

    }

    return valid;

}

int main(int argc, char \*argv[])

{

    if(argc == 2) {

        char\* text = argv[1];

        int tailleText = strlen(text);

        int pos = 0;

        // Valide ou non

        if (test(text,tailleText,&pos) == 1) {

            printf("Le text est valide.");

        } else {

            printf("Le text n'est pas valide.");

        }

    }

    return 0;

}