Practica 1 (recursividad)

R1.

Es un método, que permite dar solución a un problema mediante la invocación de sí misma. Esta permite resolver un problema subdividiendo a este en subproblemas del mismo tipo con el objetivo de que estás partes sean más simples de resolver para luego unirlas y dar solución a nuestro problema original.

R2.

Ventajas:

* Solución de algoritmos de manera más compacta.
* Elegancia en el código.
* Facilidad de verificar y comprobar que la solución es correcta.

Desventajas:

* Menos eficiente que soluciones iterativas, en términos de tiempo de computadora.
* Repetición de cálculos innecesarios.
* Desgaste de memoria en la pila y volcamiento de la misma.

R3.

Para cada llamada recursiva, se creará un nivel diferente en la pila. La función tendrá las mismas variables locales y parámetros pero con valores distintos que tienen posiciones de memorias distintas por cada ambiente recursivo que se cree.

R4.

Si, debido a que la recursividad se puede utilizar como alternativa a la iteración. La recursividad es menos eficiente en términos de tiempo de computadora, ya que; los cálculos repetitivos innecesarios ocasionan desgastes de memoria amplios.

R5.

* Tiempo de ejecución.

EJERCICIO 6

Function Factorial (integer iN): integer

if (iN == 0 OR iN == 1) then

return 1;

else

return iN \* Factorial (iN – 1);

void Combinatoria (integer iN, integer iM): integer

integer iResul

iResul = Factorial (iN)/ (Factorial (iM) \* Factorial (iN – iM));

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 2 | 8 | 8 | 9 | 10 | 11 | 6 | 5 |

Pos 0 pos 1 pos 2 pos 3 pos 4 pos 5 pos 6 pos 7

EJERCICIO 7

int iMayor = A[0], iMenor = A[0], iA2[]

function Elementos (integer iN, integer iA[], integer iIndice): bool

if (iIndice == iN) then

return true;

else

if (Mayor < A[indice]) then

Mayor = A[indice];

else

menor = A[indice];

if (iA[] % 2 == 0) then

swap iA2[] = iA[];

End

End

End

Elementos (iN, iA[ ], indice + 1)

return false;

End

EJERCICIO 8

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| s | a | l | a | s |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| r | a | l | l | a | r |

Function Capicua (integer iLi, integer iLs, char cArray[])

if (iLi >= iLs)

return true;

else

return (cArray[iLi] == cArray[iLs]) and Capicua (iLi + 1, iLs – 1)

End

End

EJERCICIO 9

122 -> no es palíndromo

1234 -> no es palíndromo

131 -> es palíndromo

1441 -> es palíndromo

101|10

01 1

1

131|10

31 13

1

Function Palindromo (integer iN, integer iM)

EJERCICIO 10

Function MCD (integer iN, integer iM): integer

if (a == b) then

return a;

if (a > b) then

return MCD (iN - iM, iM);

if (a < b) then

return MCD (iN, iM – iN);

End

EJERCICIO 11

Function Fraccion (integer iN): integer

if (iN == 1) then

return 1;

else

return 1 + Fraccion (1/1+

EJERCICIO 14

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 2 | 7 | 10 | 12 | 15 | 20 |

0 1 2 3 4 5

bool Function BinaryS (integer Array A [], integer iIndice, integer iIndice2, integer iN)

integer iMedio;

iM = (indice + indice2)/ 2;

if (A[iMedio]!= iN and iIndice == iIndice2) then

return false;

if (A[iMedio] == iN) then

return true;

if (A[iMedio] < iN) then

return BinaryS (A, iM + 1, iIndice2, iN)

if (A[iMedio] > iN) then

return BinaryS (A, indice, iM – 1, iN)