Recursividad - rtyuio

Fundamentos de la Gerencia (Universidad Peruana de Ciencias Aplicadas)

Ejercicios de Recursividad

Ejercicio 1:
Escriba una definición recursiva de una función que tiene un parámetro n de tipo entero y que devuelve el n-ésimo número de Fibonacci. Los números de Fibonacci se definen de la siguiente manera:
$F_0 = 1$
$F_1 = 1$
$F_{i+2} = F_i + F_{i+1}$

```
#include <iostream>
using namespace std;
int fibonacci(int n) {
   if (n <= 0) {
      return 0;
   }
   else if (n == 1) {
      return 1;
   }
   else {
      return fibonacci(n - 1) + fibonacci(n - 2);
   }
}

int main() {
   int n;
   cout << "Ingrese el valor de n: ";
   cin >> n;

   if (n < 0) {
      cout << "N debe ser un número no negativo." << endl;
   }
   else {
      int resultado = fibonacci(n);
      cout << "El " << n << "-ésimo número de Fibonacci es: " << resultado << endl;
   }

   return 0;
}
```

Ejercicio 2
La forma para calcular cuantas maneras diferentes tengo para elegir r cosas distintas de un conjunto de n cosas es:
C(n,r) = n! (r!*(n-r)!)
Donde la función factorial se define como
n! = n *(n-1)*(n-2)*…*2*1
Descubra una versión recursiva de la fórmula anterior y escriba una función recursiva que calcule el valor de dicha fórmula.

```
#include <iostream>
using namespace std;
// Función recursiva para calcular el coeficiente binomial C(n, r)
int coeficienteBinomial(int n, int r) {
   if (r == 0 || r == n) {
      return 1;
   }
   else {
      return coeficienteBinomial(n - 1, r - 1) + coeficienteBinomial(n - 1, r);
```

```
    }
}

int main() {
    int n, r;
    cout << "Ingrese el valor de n: ";
    cin >> n;
    cout << "Ingrese el valor de r: ";
    cin >> r;

    if (n < 0 || r < 0 || r > n) {
        cout << "Los valores de n y r deben ser no negativos y r no puede ser mayor que
n." << endl;
    }
    else {
        int resultado = coeficienteBinomial(n, r);
        cout << "C(" << n << ", " << r << ") = " << resultado << endl;
    }

    return 0;
}
```

Ejercicio 3

Escriba una función recursiva que ordene de menor a mayor un arreglo de enteros basándose en la siguiente idea: coloque el elemento más pequeño en la primera ubicación, y luego ordene el resto del arreglo con una llamada recursiva.

```
#include <iostream>
using namespace std;
void intercambiar(int& a, int& b) {
    int temp = a;
    a = b;
    b = temp;
}

// Función recursiva para ordenar un arreglo de enteros de menor a mayor
void ordenarArreglo(int arreglo[], int n) {
    if (n <= 1) {
        return; // Caso base: un arreglo de 1 elemento o menos ya está ordenado.
    }

    // Encontrar el índice del elemento mínimo en el arreglo
    int indiceMinimo = 0;
    for (int i = 1; i < n; i++) {
        if (arreglo[i] < arreglo[indiceMinimo]) {
            indiceMinimo = i;
        }
    }

    // Colocar el elemento mínimo en la primera posición
    intercambiar(arreglo[0], arreglo[indiceMinimo]);

    // Llamar recursivamente para ordenar el resto del arreglo
    ordenarArreglo(arreglo + 1, n - 1);
}

int main() {
    int arreglo[] = { 5, 2, 9, 1, 5, 6 };
    int n = sizeof(arreglo) / sizeof(arreglo[0]);

    cout << "Arreglo antes de ordenar:" << endl;
```

```
    for (int i = 0; i < n; i++) {
        cout << arreglo[i] << " ";
    }
    cout << endl;

    ordenarArreglo(arreglo, n);

    cout << "Arreglo ordenado de menor a mayor:" << endl;
    for (int i = 0; i < n; i++) {
        cout << arreglo[i] << " ";
    }
    cout << endl;

    return 0;
}
```

Ejercicio 4

Escribir una función recursiva que devuelva la suma de los primeros N enteros

```
#include <iostream>
using namespace std;
int sumaEnteros(int n) {
    if (n == 1) {
        return 1;  // Caso base: La suma de los primeros 1 entero es 1.
    }
    else {
        return n + sumaEnteros(n - 1);  // Llamada recursiva para sumar los primeros (n-1)
enteros.
    }
}

int main() {
    int N;
    cout << "Ingrese un valor para N: ";
    cin >> N;

    if (N < 1) {
        cout << "N debe ser un entero positivo o igual a 1." << endl;
    }
    else {
        int resultado = sumaEnteros(N);
        cout << "La suma de los primeros " << N << " enteros es: " << resultado <<
endl;
    }

    return 0;
}
```

Ejercicio 5

Escribir un programa que encuentre la suma de los enteros positivos pares desde N hasta 2. Chequear que si N es impar se imprima un mensaje de error.

```
#include <iostream>
using namespace std;
int main() {
    int N;
    cout << "Ingrese un valor para N: ";
    cin >> N;

    if (N % 2 != 0) {
        cout << "Error: N debe ser un número par." << endl;
```

```
    }
    else if (N < 2) {
        cout << "Error: N debe ser mayor o igual a 2." << endl;
    }
    else {
        int suma = 0;
        for (int i = N; i >= 2; i -= 2) {
            suma += i;
        }

        cout << "La suma de los enteros positivos pares desde " << N << " hasta 2 es: "
<< suma << endl;
    }

    return 0;
}
```

Ejercicio 6

Escribir un programa que calcule el máximo común divisor (MCD) de dos enteros positivos. Si M >= N una función recursiva para MCD es
MCD = M si N =0
MCD = MCD (N, M mod N) si N <> 0
El programa le debe permitir al usuario ingresar los valores para M y N desde la consola. Una función recursiva es entonces llamada para calcular el MCD. El programa entonces imprime el valor para el MCD. Si el usuario ingresa un valor para M que es < que N el programa es responsable de switchear los valores.

```
#include <iostream>
using namespace std;
// Función recursiva para calcular el MCD
int calcularMCD(int M, int N) {
    if (N == 0) {
        return M;
    }
    else {
        return calcularMCD(N, M % N);
    }
}

int main() {
    int M, N;
    cout << "Ingrese el valor para M: ";
    cin >> M;
    cout << "Ingrese el valor para N: ";
    cin >> N;

    if (M < N) {
        // Si M es menor que N, intercambiamos los valores
        int temp = M;
        M = N;
        N = temp;
    }

    if (M <= 0 || N <= 0) {
        cout << "M y N deben ser enteros positivos." << endl;
    }
    else {
        int mcd = calcularMCD(M, N);
        cout << "El MCD de " << M << " y " << N << " es: " << mcd << endl;
```

```
    }

    return 0;
}
```

Ejercicio 7

Programe un método recursivo que transforme un número entero positivo a notación binaria.

```
#include <iostream>
using namespace std;
void decimalABinario(int n) {
    if (n > 0) {
        decimalABinario(n / 2);  // Llamada recursiva con la parte entera de la división por 2
        cout << n % 2;  // Imprime el residuo de la división por 2 (el bit actual)
    }
}

int main() {
    int numero;
    cout << "Ingrese un número entero positivo: ";
    cin >> numero;

    if (numero <= 0) {
       cout << "El número debe ser un entero positivo." << endl;
    }
    else {
        cout << "Notación binaria de " << numero << " es: ";
        decimalABinario(numero);
       cout <<endl;
    }

    return 0;
}
```

Ejercicio 8

Programe un método recursivo que transforme un número expresado en notación binaria a un número entero.

```
#include <iostream>
#include <cmath>
using namespace std;
int binarioAEntero(std::string binario, int indice) {
    // Caso base: Si hemos recorrido todos los dígitos, retornar 0.
    if (indice < 0) {
        return 0;
    }

    // Obtener el dígito actual (0 o 1) como carácter y convertirlo a entero.
    int digito = binario[indice] - '0';

    // Realizar la llamada recursiva para los dígitos restantes y sumar el valor actual,
    // multiplicado por 2 elevado a la posición del dígito.
    return digito * pow(2, binario.length() - 1 - indice) + binarioAEntero(binario, indice -
1);
}

int main() {
    string binario;
    cout << "Ingrese un número en notación binaria: ";
```

```
    cin >> binario;

    bool esBinarioValido = true;

    for (char c : binario) {
        if (c != '0' && c != '1') {
            esBinarioValido = false;
            break;
        }
    }

    if (!esBinarioValido) {
        cout << "El número ingresado no está en notación binaria válida." << endl;
    }
    else {
        int entero = binarioAEntero(binario, binario.length() - 1);
        cout << "El número en notación binaria " << binario << " es igual a " << entero
<< " en decimal." << endl;
    }

    return 0;
}
```

Ejercicio 9

Programe un método recursivo que calcule la suma de un arreglo de números enteros.

```
#include <iostream>
using namespace std;
int sumaArreglo(int arreglo[], int n) {
    // Caso base: si el arreglo está vacío, la suma es 0.
    if (n == 0) {
        return 0;
    }
    else {
        // Llamada recursiva para sumar los elementos restantes del arreglo.
        return arreglo[n - 1] + sumaArreglo(arreglo, n - 1);
    }
}

int main() {
    int n;
    cout << "Ingrese la longitud del arreglo: ";
    cin >> n;

    if (n < 1) {
        cout << "La longitud del arreglo debe ser al menos 1." << endl;
    }
    else {
        int arreglo[n];
        cout << "Ingrese los elementos del arreglo:" << endl;
        for (int i = 0; i < n; i++) {
            cin >> arreglo[i];
        }

        int resultado = sumaArreglo(arreglo, n);
        cout << "La suma de los elementos del arreglo es: " << resultado << endl;
    }

    return 0;
}
```

Ejercicio 10

Programe un método recursivo que invierta los números de un arreglo de enteros.

```
#include <iostream>
using namespace std;
void invertirArreglo(int arreglo[], int inicio, int fin) {
   // Caso base: cuando el índice de inicio supera al índice de fin, no hay más elementos
para intercambiar.
   if (inicio >= fin) {
      return;
   }

   // Intercambiar el elemento en el índice de inicio con el elemento en el índice de fin.
   int temp = arreglo[inicio];
   arreglo[inicio] = arreglo[fin];
   arreglo[fin] = temp;

   // Llamada recursiva para invertir el resto del arreglo, excluyendo los elementos ya
invertidos.
   invertirArreglo(arreglo, inicio + 1, fin - 1);
}

int main() {
   int n;
   cout << "Ingrese la longitud del arreglo: ";
   cin >> n;

   if (n < 1) {
      cout << "La longitud del arreglo debe ser al menos 1." << endl;
   }
   else {
      int arreglo[n];
      cout << "Ingrese los elementos del arreglo:" << endl;
      for (int i = 0; i < n; i++) {
         cin >> arreglo[i];
      }

      invertirArreglo(arreglo, 0, n - 1);

      cout << "El arreglo invertido es:" << endl;
      for (int i = 0; i < n; i++) {
         cout << arreglo[i] << " ";
      }
      cout << endl;
   }

   return 0;
}
```

Ejercicio 11

Cuál es el resultado de esta función para distintos valores de X?

```
Static int f(int x)
{
        if (x >100)
        {
                return (x-10);
```

```
        }
        else
        {
                return(f(f(x+11)));
        }
}
```

```
#include <iostream>
using namespace std;
int f(int x) {
   if (x > 100) {
      return (x - 10);
   }
   else {
      return f(f(x + 11));
   }
}

int main() {
   int x;
   cout << "Ingrese un valor para x: ";
   cin >> x;

   int resultado = f(x);

   cout << "El resultado de f(" << x << ") es: " << resultado << endl;

   return 0;
}
```

Ejercicio 12

Implemente una función recursiva que nos diga si una cadena es palíndromo.

```
#include <iostream>
#include <string>
#include <cctype> // Necesario para la función tolower
using namespace std;
bool esPalindromo(std::string cadena) {
   // Caso base: Si la cadena es de longitud 0 o 1, es un palíndromo.
   if (cadena.length() <= 1) {
      return true;
   }

   cadena.erase(remove_if(cadena.begin(),cadena.end(),isspace),cadena.end());
   transform(cadena.begin(),cadena.end(),cadena.begin(),tolower);

   // Comprobar si el primer y último carácter de la cadena son iguales.
   if (cadena[0] == cadena[cadena.length() - 1]) {
      // Llamada recursiva con la cadena sin el primer y último carácter.
      return esPalindromo(cadena.substr(1, cadena.length() - 2));
   }

   // Si los primeros y últimos caracteres no son iguales, no es un palíndromo.
   return false;
}

int main() {
   string cadena;
```

```
    cout << "Ingrese una cadena: ";
    getline(cin, cadena);

    if (esPalindromo(cadena)) {
        cout << "La cadena es un palíndromo." << endl;
    }
    else {
        cout << "La cadena no es un palíndromo." << endl;
    }

    return 0;
}
```

Ejercicio 13

Diseñe e implemente un algoritmo que imprima todas las posibles descomposiciones de un número natural como suma de números menores que él.

1= 1
2 = 1+1
3= 2 + 1
3= 1+1+1
4= 3+1
4= 2+1+1
4 = 1+1+1+1
4=2+2
4=2+1+1
4=1+1+1+1
N = (n-1) +1

```
#include <iostream>
#include <vector>
using namespace std;
void descomposiciones(int n, vector<int>& descomposicion_actual) {
    if (n == 0) {
        // Hemos encontrado una descomposición completa, la imprimimos.
        for (size_t i = 0; i < descomposicion_actual.size(); i++) {
            cout << descomposicion_actual[i];
            if (i != descomposicion_actual.size() - 1) {
                cout << " + ";
            }
        }
        cout << endl;
    }
    else {
        // Generamos las descomposiciones utilizando números menores que n.
        for (int i = 1; i <= n; i++) {
            if (descomposicion_actual.empty() || i <= descomposicion_actual.back()) {
                // Evitamos duplicados y mantenemos el orden decreciente.
                descomposicion_actual.push_back(i);
                descomposiciones(n - i, descomposicion_actual);
                descomposicion_actual.pop_back();
            }
        }
    }
}

int main() {
    int n;
    cout << "Ingrese un número natural: ";
```

```
    cin >> n;

    if (n <= 0) {
        cout << "El número debe ser un número natural (mayor que 0)." << endl;
    }
    else {
        vector<int> descomposicion_actual;
        cout << "Descomposiciones de " << n << " como suma de números menores que
él:" << endl;
        descomposiciones(n, descomposicion_actual);
    }

    return 0;
}
```

$$N = (n-2) + 2 = (n-2) + 1 + 1$$