1. Considera el algoritmo denominamos β mostrando abajo
Indica qué hace el algoritmo β y demuestra que es correcto con respecto a la precondición y postcondición dadas.

```
Procedure B (A:Array ; a, b, x: entero)
    // PreCond: a  b y x  A[a..b]
    var i : enteros;
    Begin
        i ← a ;
        While x 6= A[i] do
            i ← i + 1;
        Return i
    End
    // PostCond: a  i  b y x 6 A[a..i  1] y x = A[i]
```

Respuesta
El algoritmo escoge una x arbitraria y esta itera sobre el arreglo, hasta que encuentre un índice el cual cumpla la condicón de: x = A[i] y regresa dicho ındice.

Después de que el algoritmp encuentra un índice, mientras el arreglo contiene el entero x que fue solicitado. Ahora demostraremos un funcionamiento correcto:

Si a es tal que x = A[a], el for nunca se cumplíra y se regresa a, ahora supongamos que x = A[k], con a ¡ x b, esto es menor o igual a b por la preCond.

Suponemos que no hay elementos repetidos y A[k] 6 = A[j], para toda j a...b k. Entonces la función hace exactamente k a iteraciones. En el caso de haber repeticiones el número de iteraciones puede o no ser menor a este.

En cualquiera de estos dos casos siempre se tendremos un índice en [a, ..b], pues x se encuentra en este rango.

Por otro lado x 6 ∈ A[a...i 1], pues en dicho caso se regresaria el entero en dicho rango que cumpla con ser x. Por último siempre se cumple que x = A[i], pues es lo que nos permite salir del ciclo, además de que y no es infinito por la caracteristica partıcular de x.

2. Problema Evaluación de Polinomio. El Algoritmo Horner evalua, en el punto $x = x_o$ el polinomio

$$P_n(x) = a_0 + a_1x + a_2x^2 + ... + a_nx^n$$

Usando la Técnica del Invariante del ciclo, demuestra que el algoritmo dado es correcto.

```
Procedure Horner (a:Pol ; x0: entero)
    var i, total : enteros;
    Begin
        total ← 0 ;
        For i = k  1 to 0 by 1 do
            total ← a[i] + total × x0
        Return total
    End
```

Respuesta
Proponemos como invariante a k, el resultado final tiene a un polinomio de grado a lo mas k 1. Donde las constantes son los elementos del arreglo como k = 1: total = a[n] · xO + a[n 1], se cumple.

Asumimos que para alguna k es verdadera, ahora veremos esta se cumple para k + 1. Recordando la definición de Horner, tenemos que el: total = a[n k] + total x_0

Entonces por hipótesis de inducción este ya era un polinomio de grado a lo más k y al ser evaluado por su constante, en el caso de ser distinto de a 0, el polinomio es evaluado en k + 1, por ende este puede ser una invariante.

En otro caso con la negación del ciclo tendriamos que i = 1, en el caso de que sea iterado el arreglo y se hayan hecho las n evaluaciónes de a también, entonces el ciclo no seria infinito, ya que, n-1 se va a 0.

3. Caluclar la suma de los primeros n números impares

- 3.a) Proporciona un algoritmo iterativo (pseudo-código) que solucione el Problema .
Respuesta
Esta es una implementacion en Java, tenemos como **PreCond** n, es un entero mayor a cero y **PostCond** en sum tenemos la suma de los primeros n impares.

```
public int sumaImpares(int n){
    int i, sum;
    sum = 0;
    for(i = 1; i < = n; i ++){
        sum += 2*i-1;
    }
    return sum;
}
```

- 3.b) Demuestra que tu algoritmo es correcto, usando la técnica del Invariante del ciclo.
Respuesta
Proponemos como invariante que para la i-esima iteración en suma se encuentra la suma de los primeros $i-1$ números impares.
Ahora haremos inducción sobre i supongamos que $i = 1$, sum $= 0$ y almacena la suma de los primeros números impares $1-1 = 0$. Supongamos que para alguna $i < n-2$ se cumple la invariante. Notaremos que se cumple para $i+1$ en la $i+1-$ iteración.
Recordando nuestra hipostesis de inducción que dice que, a sum tiene la suma de los primeros i números impares y le sumamos $2*(i+1)-1$, que como ya sabemos $2*(i+1)-1$ es el $(i+1)-$ número impar.
Antes de la iteración $i+2$ se encuentra en sum , es decir en la suma de los primeros $i+1$ números impares, con esto obteniendo la invariante correcta. Ahora negando el ciclo, la invariante implicara la postcondición que se muestra acontinuación:

$$sum = \sum_{j=1}^{i-1} 2j - 1 \land i = n+1 \rightarrow sum = \sum_{j=1}^{n} 2j - 1$$

Con sustitución se puede ver que es cierta, ya que la negación del ciclo es cuando $i = n+1$ y no estamos iterados arreglos y el ciclo no se vuelve infinito debido a que a i nunca la modificamos en el cuerpo del for.

- 4.a) Proporciona un algoritmo recursivo (pseudo-código) que solucione el Problema .
Respuesta
Esta la implementacion en Haskell, tenemos como **PreCond** El parametro debe ser mayor a cero y debe de ser un entero y como **PostCond** x es el n-esimo entero positivo y este debe regresar la suma acumulada de las llamdas

```
impares :: Integer -> Integer
impares 1 = 1
impares x = (2* x -1) + sum_impares (x -1)
```

- 4.b) Demuestra que tu algoritmo es correcto, usando inducción matemática.
Respuesta
Ahora haremos inducción sobre x, supongamos que x=1 y de acuerdo a nuestro código este regresa 1 y significa que es correcto. Después supongamos que $x = n$ se cumple que regresa los primeros n números impares, entonces para $x = n+1$, sabemos que la n-esima llamada recursiva lleva la suma de los n impares y viendo nuestro caso recursico que es $2(n+1) + impares(x-1)$, este nos regresara el $n+1$-iesimo números par y al terminar la n-esima llamada del programa se volvera a llamar con el valor de 1, pues recordando la primera llamada era $n+1$ y este recae en el primer caso que es 1 y regresa 1.

- 5.(Opcional) Considera el proceso SSort(A,a,b), descrito abajo. Demuestra usando la técnica del Invariante del ciclo que el algoritmo es correcto.

```
(*Pre: a < b + 1 *)
    SSort(A,a,b);
(*Post: A[a]  A[a + 1]  . . .  A[b] *)
```

```
Procedure SSort(var A:Atype; a,b:integer):
    var i:integer;
    begin
        if a = b+1 then Do_Nothing
        Else
            i <- MinIndex(A,a,b);
            if i != a then Swap(A[i],A[a]);
            SSort(A,a+1,b)
    end; {SSort}
```

Suponer que MinIndex(A,i,j) regresa el ındice del elemento mínimo de un arreglo no vacío A[i..j] y Swap(A[i],A[j]) intercambia los dos elementos indicados.

Respuesta

Proponemos como invariante $n-1$ para el indice regrese el elemento mínimo de un arreglo vacío. Sabemos que todos los elementos del arreglo son mayores a a, además de que no tendremos que recorrer todos los elementos, ya que cuando i valga $(n-1)$ todos los elementos a su izquierda estarán ordenados y serán menores que él.

Ahora para que el arreglo pueda aceptar el vació tenemos que la condición debe de ser $i >= n-1$, ya que con esta condición si el arreglo es vació este no entrara al loop. Entonces si $i = 0$ incluye al vació, luego vamos a buscar en el loop el menor elemento que se encuentre $a[0, 1, 2, ..., i-1]$, como fue mencionado hace rato sabemos que todo los elementos son menores que él $a[i, ..., n-1]$ entonces con esto podemos intercambiar la posición del menor elementos en $a[i]$ y aumentar el índice de i, por consiguiente al tomar al menor elemento de la sección desordenada y pasarlo a la sección ordenada de los restantes, siendo estos mayores que cualquier elemento en el arreglo ya ordenado.