Algoritmo iterativo

Total de pontos 0/100

```
Quadrado( x ) : natural
Entrada: x>0 e natural.
Saída : x^2.
1)  i = 1;
2) q = 1;
3) enquanto ( i != x) faça
4)       q = q + 2*i + 1;
5)        i = i + 1;
6) fim;
7) retorne q;
8) Fim.
```

O e-mail do participante (**ilrocha.cic@uesc.br**) foi registrado durante o envio deste formulário.

0 de 60 pontos

✕ Faça um rascunho para identificar o invariante: * .../10

A condição para o loop terminar é:
i = x

Quando o loop é encerrado, queremos que
Q = 1, 4, 9, 16, ..., x*x

Substituindo o valor "i = x" em "Q", temos:
Q = i*i
Que é um possível candidato a invariante

Feedback

Rascunho:
O enquanto termina quando i=x.
Na linha 7 queremos que q=x².
Substituindo: q=i² -> candidato a invariante.

✕ Enuncie o invariante: * .../10

Seja {j ∈ N; j > 0}, vamos mostrar a corretude da repetição mostrando que antes de cada iteração "j" na linha 3, vale o seguinte invariante:
Q_j = i_j * i_j

Feedback

Vamos provar a corretude da função quadrado(x), mostrando que antes de cada iteração i do enquanto vale o seguinte invariante:
q_i=i².

✕ Prove a inicialização: * .../10

Antes da primeira iteração, o invariante deve ser
Q_1 = i_1 * i_1

Na linha 1 e 2, temos que
i_1 = 1

Assim, o invariante é dado por:
Q_1 = 1 * 1
Q_1 = 1

O que está correto, pois o quadrado de 1 é igual a 1

Feedback

Antes da primeira iteração, das linhas 1 e 2 temos que:
i =1 e q _1 = 1, valor esse que está de acordo com o invariante:
q _1 = i²= 1² = 1.
Portanto o invariante vale antes da primeira iteração.

✕ Prove a manutenção: * .../20

(H.I) Suponha uma iteração k, onde {k ∈ N; k > 0}, e que o seguinte invariante é verdadeiro:
Q_k = i_k * i_k

Mostrar que o invariante vale antes de uma iteração "k + 1", ou seja:
Q_(k+1) = i_(k+1) * i_(k+1)

Para isso, vamos executar a iteração k:
Na linha 4:
Q_(k+1) = Q_k + 2*i_k + 1
Da HI temos que:
Q_k = i_k * i_k
Substituindo, temos:
(a) -> Q_(k+1) = (i_k * i_k) + 2*i_k + 1
Simplificando:
(a) -> Q_(k+1) = (i_k + 1) * (i_k + 1)
Na linha 5 temos:
i_(k+1) = i_k + 1
Substituindo em (a), temos:
Q_(k+1) = [i_(k+1)]*[i_(k+1)] , c.q.d.

Portanto conseguimos provar que o invariante segue a Hipótese desejada.

Feedback

Suponha que o invariante vale antes de uma iteração i>=1, ou seja q_i = i².
Vamos provar que o invariante vale antes da iteração i + 1, ou seja:
*q_(i+1) = (i+1)² = i² + 2*i + 1.*
Para mostrar, vamos executar a iteração i:
Na linha 4 da iteração i, temos que:
q_(i+1) = q_i + 2 ∗ i + 1;. Da hipótese vamos substituir q_i:
q_(i+1) = i² + 2 ∗ i + 1, c.q.d.

Portanto o invariante vale antes de qualquer iteração do enquanto.

✕ Término: * .../10

No final do laço, onde "i = x", foi provado que o invariante:
Q = i * i

Assim, substituindo "i = x" no invariante, temos:
Q = x * x, c.q.d

Portanto o algoritmo devolve "x * x", que é o quadrado de X.

Feedback

O enquanto termina quando i=x.
Substituindo no invariante q = i² temos que:
q = x², ou seja a função devolve o que promete.

Algoritmo recursivo

0 de 40 pontos

```
Potencia( x, n)
Entrada: x natural maior que zero e n natural.
Saída: x^n.
1) se (n==0) devolva 1;
2) devolva x*Potencia(x, n-1);
Fim.
```

✕ Enuncie a demonstração. * .../5

Vamos provar por indução em n, que o algoritmo calcula "x^n" para x inteiro maior que 0 e n natural.

Feedback

Vamos provar por indução em n que Potencia(x,n) devolve x^n.

✕ Prove a base: * .../5

Para n == 0, o algoritmo devolve 1 na linha 1, o que está correto, pois:
x^0 = 1

Feedback

Para n=0 temos que a chamada Potencia(x,0) executa a linha devolvendo 1, o que está correto uma vez que x^0=1.

✕ Enuncie a hipótese de indução: * .../10

(H.I.) Supondo que para um "n >= 0" natural a chamada da função Potencia(x, n) devolve: x^n

Feedback

Suponha que para um n>=0 a chamada Potencia(x,n) devolve x^n.

✕ Desenrole o passo. * .../20

Vamos mostrar que para "n+1" a chamada da função Produto(x, n+1) devolve "x^(n+1)"

Para isso vamos executar a chamada: Produto(x, n+1)

Como "n+1 > 0", a condição de parada é falsa, e a linha 2 é executada:
return x * Potencia(x, [n+1]-1);
Simplificando, temos:
return x * Potencia(x, n);

Pela H.I., sabemos que a chamada Potencia(x, n) devolve x^n. Portanto, substituindo:
return x * (x^n);
Simplificando, temos:
return x^(n+1); c.q.d.

Portanto, pelo princípio de indução matemática, a função devolve o que promete.

Feedback

Vamos mostrar que para n+1 a chamada Potencia(x,n+1) devolve x^(n+1).

Para isso vamos executar a chamada Potencia(x,n+1).
Como n+1>0 a linha 2 será executada:
*devolva x*Potencia(x, (n+1)-1);*
*devolva x*Potencia(x, n+1-1);*
*devolva x*Potencia(x, n); (a)*
Pela HI sabemos que Potencia(x,n)=x^n. Substituindo em (a):
*devolva x*x^n;*
devolva x^(n+1); c.q.d

Portanto, pelo PIM, o algoritmo devolve o que promete.