Ejercicio 2

(a)

El invariante correcto es el siguiente:
$\mathcal{I}_2 : (1 \leq i \leq N+1) \land (res = \sum_{j=0}^{i-1} j^2)$

- Notar que i terminará valiendo uno más que N, por eso digo que va hasta N+1.
- El índice de la sumatoria es j, que funciona de manera análoga con el índice propuesto i. Cambiamos de nombre de índice para que no haya confusiones.

(b)

Precondición $P_c : (N > 0) \land (i = 1) \land (res = 0)$

- Hay que incluir también la precondición de la función en sí. Todo lo que sepas antes del ciclo, hay que incluirlo.

Postcondición $Q_c : (i = N+1) \land (res = \sum_{i=1}^{N} i^2)$

- i terminará valiendo N+1 porque hay que contar la iteración en la cual la guarda se vuelve falsa.
- En la postcondición, se debe explicar el proceso del loop en términos de la guarda (en este caso, N).

(c)

$i = 3, N = 3, res = 5$:

- Cumple el invariante, porque $(1 \leq 3 \leq 3+1) \land (5 = \sum_{j=0}^{3-1} j^2)$
- Cumple la guarda del ciclo, porque $3 \leq 3$.

(d)

- $(i = 3) \land (N = 3) \land (res = 5)$
- $(i = 3) \land (N = 3) \land (res = 5 + 3 \cdot 3)$
- $(i = 4) \land (N = 3) \land (res = 14)$

(e)

$$P_c \Rightarrow \mathcal{I}$$

$$(N > 0) \land (i = 1) \land (res = 0) \Rightarrow (1 \leq i \leq N + 1) \land (res = \sum_{j=0}^{i-1} j^2)$$

Como N es mayor que cero, se puede asumir que $N \geq 1$. Por lo que podemos cambiar eso.

$$(N \geq 1) \land (i = 1) \land (res = 0) \Rightarrow (1 \leq i \leq N + 1) \land (res = \sum_{j=0}^{i-1} j^2)$$

Podemos ver que se cumple el primer término de $\mathcal{I}$, porque $1 \leq 1 \leq 1 + 1$ es verdadero. También se cumple el segundo término, porque $0 = \sum_{j=0}^{1-1} j^2$. Se cumple ya que si el límite de la sumatoria es cero, la sumatoria en sí da cero, por lo que $0 = 0$ es verdadero.

(f)

$$\neg B \land \mathcal{I} \Rightarrow Q_c$$

$$(i > N) \land \left((1 \leq i \leq N + 1) \land (res = \sum_{j=0}^{i-1} j^2) \right) \Rightarrow (i = N + 1)$$

- $(i = N + 1)$ vale, porque por $\mathcal{I}$ teníamos que $(1 \leq i \leq N + 1)$.

Ejercicio 3

(a)

```cpp
bool todos_positivos(const vector<int> & vec)
{
    // assuming vec is not empty
    int i = 0;
    while (i < vec.size())
    {
        if(vec[i] < 0)
        {
            // only one negative element is enough to return false
            return false;
        }
        i++;
    }
    return true;
}
```

- $P_c : (|vec| \geq 1) \wedge (i = 0)$
- Q_c: *if* $\left(\sum_{i=0}^{|vec|-1} \beta(esPositivo(vec[i])) = |vec| - 1\right)$ *then* res = True *else* res = False *fi* $\wedge (i = |vec|)$

  esPositivo (i:int) $\equiv i > 0$
- $\mathcal{I} : (0 \leq i \leq |vec|) \wedge \left(res = True \Leftrightarrow \sum_{j=0}^{i-1} \beta(esPositivo(vec[j])) = |vec| - 1\right)$

(b)

```
bool no_contiene_equis(const vector<char> & str)
{
    // assuming str is not empty
    int i = 0;
    while(i < str.size())
    {
        if(str[i] == 'x')
        {
            // soon as one 'x' shows up it is false
            return false;
        }
        i++;
    }
    return true;
}
```

- $P_c : (|str| \geq 1) \wedge (i = 0)$
- Q_c :*if* $(1 \leq \sum_{i=0}^{|str|-1} \beta(\text{esX (str[i])}) \leq |str| - 1)$ *then* (res = False) *else* (res = True) *fi* $\wedge (i = |str|)$

  esX (c:char) $\equiv c =' x'$

  Si la cantidad de ocurrencias de 'x' está entre 1 y el tamaño de str, quiere decir que sí apareció una 'x', por lo que es falso.

Ejercicio 4

- $P_c : (i \geq 0) \wedge (n \geq i) \wedge (res \geq 1)$
- $Q_c : (res = \prod_{i=0}^{N} i) \wedge (i = n + 1)$
- $\mathcal{I} : (0 \leq i \leq n + 1) \wedge (res = \prod_{j=0}^{i-1} j)$

Ejercicio 5

(a)

- $P_c : (N > 0) \land (i = 1) \land (res = 2)$
- $Q_c : (i = N) \land (res = \sum_{i=1}^{N-1} \beta(N \bmod i = 0))$

(b)

El invariante correcto es $\mathcal{I}_3$.

(c)

- $i = 1, N = 4$
- Cumple que $1 \leq 2 < 4$ y que $2 = 2 * 1$

(d)

Ejercicio 6

(a)

```
void sumar_n(vector<int> & ls, const int & n)
{
    // init iteration var
    int i = 0;

    // iterate over ls
    while(i < ls.size())
    {
        // modify
        ls[i] = ls[i] + n;
        i++;
    }
}
```

- $P_c : (i = 0)$
  Sólo la variable de iteración. No es necesario agregar nada más, ni siquiera alguna precondición del programa.
- $Q_c : (|ls_0| = |ls|) \land (i = |ls|) \land (\forall i{:}\text{int})(0 \leq i \leq |ls| \Rightarrow ls[i] = ls_0[i] + n)$
  El tamaño del vector inicial se mantiene luego de modificar sus elementos. i terminará valiendo $|ls|$ porque se debe incumplir la guarda en algún momento. Todos los elementos de ls serán los elementos de ls_0 sumado n.

- $\mathcal{I} : (0 \leq i \leq |ls|) \land (\forall j{:}\text{int})(0 \leq j < i \Rightarrow ls[j] = ls_0[j] + n)$
  i parte desde cero por la precondición y terminará valiendo lo que el tamaño de ls. Todos los valores de j entre cero e i irán modificando los valores de ls_0. Uso j para diferenciarlo de i.

(b)

```
int suma_y_reset(vector<int> & v)
{
    // init res and iteration var
    int res, i = 0;

    // iterate over v
    while(i < v.size())
    {
        // add to res
        res += v[i];
        // reset
        v[i] = 0;
        i++;
    }

    return res;
}
```

- $P_c : (i = 0) \land (res = 0)$
  Por la declaración de estas variables en cero.
- $Q_c : \left(res = \sum_{i=0}^{|v_0|-1} v_0[i]\right) \land (i = |v_0| = |v|) \land (\forall i{:}\text{int})(0 \leq i \leq |v| \Rightarrow v[i] = 0)$
  res es la sumatoria de todos los valores en v_0. i va a terminar siendo igual al tamaño de v_0, y, como no vamos a cambiar el tamaño del vector, se mantiene que $|v_0| = |v|$. Como reseteamos todos los valores, tenemos que todos los valores del vector modificado, v, serán cero.
- $\mathcal{I} : \left(res = \sum_{j=0}^{i-1} v_0[j]\right) \land (\forall j{:}\text{int})(0 \leq j < i \Rightarrow v[j] = 0) \land (0 \leq i \leq |v|)$
  res es la sumatoria desde j hasta $i - 1$. Se puede decir que j "va por atrás de i" modificando las cosas. Importante aclarar el rango de valores de i.

Ejercicio 7

```
while (i < v.size()) {
  res = res * v[i];
  i = i + 1;
}
```

- $P_c : (|v| \geq 1) \land (res \neq 0)$
  Si res fuera cero, no tendría sentido que multipliquemos todo por cero. Además, v debería tener al menos un elemento para acceder, para que no se indefina en la primera línea de la iteración.
- $Q_c : \left(res = \prod_{i=0}^{|v|-1} v[i]\right) \land (i = |v|)$
  res será la productoria de todos los elementos de v. i terminará valiendo $|v|$ porque necesitamos que el iterador termine en algún momento.
- $\mathcal{I} : \left(res = \prod_{j=0}^{i-1} v[j]\right) \land (0 \leq i \leq |v|)$ res es la productoria de todos los índices j que van "por detrás" de i. El rango de i está aclarado.

Ejercicio 8

```
int i = 0; res = 0;
while (i < v.size()) {
  res = res + v[i];
  i = i + 1;
}
res = res + 100;
```

$P_c :\ i = 0 \land res = 0 \land |v| \mod 2 = 0$
$Q_c :\ res = \sum_{i=0}^{|v|-1} v[i]$
$I :\ 0 \leq i \leq |v|/2 \land res = \sum_{j=0}^{i-1} v[2j] + v[2j+1]$

(a)

El invariante no respeta al código, porque i termina adquiriendo el valor de $|v|$ para la última iteración (esto es $0 \leq i \leq |v|$). Además, la suma de los elementos de v a res son comunes, no es que esté mal la sumatoria en sí (porque llegaríamos al mismo resultado de res), pero no es precisamente lo que hace el iterador.

(b)

Me piden:

- Vector de tamaño par.
- Inicializar las variables de iteración y la respuesta en 0.
- Cuando termine el ciclo, *res* será la sumatoria de todos sus elementos.
- La iteración llegará a la mitad del vector, y se sumarán a *res* todos los elementos de a "parejas".

```
int sumar_hasta_mitad(const vector<int> & v)
{
    int i, res = 0;
    while(i < v.size()/2)
    {
        res = res + v[2*i] + v[(2*i)+1];
        i++;
    }
    res = res + 100;
    return res;
}
```

(c)

```
vector<int> v = {1, 2, 3, 4, 5, 6};
int i = 3;
int res = 32;
```

- Cumplen el invariante porque: $0 \leq 3 \leq 6/2 = 3$ y
  $32 = \sum_{j=0}^{3-1=2} v[2j] + v[2j+1] \Rightarrow$
  $32 = (0 + v[0] + v[1]) + (3 + v[2] + v[3]) + (10 + v[3] + v[4]) \Rightarrow$
  $32 = (0 + 1 + 2) + (3 + 3 + 4) + (10 + 4 + 5) \Rightarrow$
  $32 = 3 + 10 + 19 \Rightarrow$
  $32 = 32$

(e)

- Satisface el invariante, porque $0 \leq 3 \leq 3$. Como dije arriba.
- No satisface la guarda, porque $3 < 6/2$ es falso.
- Satisface la postcondición P_c, porque ... "no entendí el código, porque según lo que entendí, suma todos los números menos el último. ¿Estará bien escrito lo de res en el código? Parece que estoy acumulando los resultados de cada iteración anterior. Yo sé que debería ser la suma de todos los elementos del vector".

(f)

Quiero demostrar que $\neg B \land \mathcal{I} \Rightarrow Q_c$
$\equiv (i \geq |v|/2) \land \left(0 \leq i \leq |v|/2 \land res = \sum_{j=0}^{i-1} v[2j] + v[2j+1]\right) \Rightarrow \left(res = \sum_{i=0}^{|v|-1} v[i]\right)$

- La postcondición vale, porque si tomamos que $i \geq |v|/2 \land i \leq |v|/2$ tenemos que $|v|/2 = i \Rightarrow |v| = 2i$. Si reemplazamos $|v|$ en el límite de la sumatoria esta se anulará porque quedaría $res = \sum_{i=0}^{2i=0} v[i] = 0$.

Ejercicio 9

Devuelve una posición del mínimo elemento de una secuencia, y aumenta el valor del elemento en esa posición.

```
int minima_posicion(vector<int>& v)
```
Pre: $v = v_0 \land |v| > 0$
Post: $0 \leq res < |v| \land$
$((\forall i : \texttt{int})\ 0 \leq i < |v| \implies v[i] \geq v[res]) \land$
$((\forall i : \texttt{int})\ 0 \leq i < |v| \land i \neq res \implies v[i] = v_0[i]) \land$
$v[res] = v_0[res] + 1$

```
int i = 0; res = 0;
while(i < v.size()) {
    if (v[i] < v[res])
        res = i;
    i = i + 1;
}
v[res] = v[res] + 1;
```

(a)

- $P_c : (v = v_0) \land (|v| > 0) \land (i = 0) \land (res = 0)$
  Incluimos las cosas de la precondición de la función, y las declaraciones de variables.
- $Q_c : (0 \leq res < |v|) \land (\forall i{:}\text{int})(0 \leq i < |v| \Rightarrow v[i] \geq v[res]) \land$
  $(\forall i{:}\text{int})(0 \leq i < |v| \land i \neq res \Rightarrow v[i] = v_0[i]) \land (i = |v|)$
  Copié las pre y postcondiciones de la función, menos la línea en la que se incrementa el mínimo elemento. Además, i terminará adquiriendo el valor de $|v|$ porque se deja de cumplir la guarda.

- $\mathcal{I} : (0 \leq i \leq |v|) \land (0 \leq res < |v|) \land (\forall j{:}\text{int})(0 \leq j < i \Rightarrow v[res] \leq v[j])$
  Defino los rangos de i y de res. Notar que res no puede ser $|v|$. Explico que todo elemento en el índice j será mayor al elemento en el índice res.

(b)

Quiero probar que $\mathcal{I} \land \neg B \Rightarrow Q_c$.
$((0 \leq i \leq |v|) \land (0 \leq res < |v|) \land (\forall j : int)(0 \leq j < i \Rightarrow v[res] \leq v[j]))$
$\land (i \geq |v|)$
$\Rightarrow (0 \leq res < |v|) \land (\forall i{:}\text{int})(0 \leq i < |v| \Rightarrow v[i] \geq v[res]) \land$
$(\forall i{:}\text{int})(0 \leq i < |v| \land i \neq res \Rightarrow v[i] = v_0[i]) \land (i = |v|)$

- Por $(i \leq |v|) \land (i \geq |v|) \Rightarrow (i = |v|)$ del invariante y la guarda negada, tenemos que es válido que $(i = |v|)$ vale en la postcondición.

Ejercicio 10

$P_c : \ i = 0 \land v = v_0$
$Q_c : \ |v| = |v_0|$
$\land (\forall k : \texttt{int})(0 \leq k < |v| \land v_0[k] \mod 2 \neq 0 \implies v[k] = v_0[k])$
$\land (\forall k : \texttt{int})(0 \leq k < |v| \land v_0[k] \mod 2 = 0 \implies v[k] = v_0[k]/2)$

```
1 while (i < v.size()) {
2   if(v[i] % 2 = 0){
3     v[i] = v[i]/2;
4   }
5   i = i + 1;
6 }
```

(a)

$\mathcal{I} : (0 \leq i \leq |v|) \land (\forall j{:}\text{int})(0 \leq j < i \land v_0[j] \bmod 2 = 0 \Rightarrow v[j] = v_0[j]/2)$
i va a llegar al valor de $|v|$, y todos los elementos en $v[j]$ pares se van a dividir por dos. *¿Es necesario aclarar qué pasa si el elemento es impar?*

(b)

Tenemos $i = 0$ y que $v_0 = \{4, 3, 1, 8\}$ con $|v_0| = 4$. Preservará el invariante porque $0 \leq 0 \leq 4$. La ejecución terminará con $|v_0| = 4$, $v = \{2, 3, 1, 8\}$ y con $i = 1$.

(c)

La función variante es $f_v : |v| - i$. Esta función es estrictamente decreciente en función de i. Si esta función llega a cero, quiere decir que $f_v = 0 \Rightarrow |v| - i = 0$. Esto se cumple si $i = |v|$.

La guarda B se hace falsa cuando $i \geq |v|$, y cuando $i = |v|$ (que es cuando la función variante se hace cero) se cumple la terminación del ciclo.

Ejercicio 11

Ejercicio 12

(a)

```
bool es_palindromo(const vector<char> & s)
{
    int j = 0;
    while(j < s.size())
    {
        if(s[j] != s[s.size()-j-1])
        {
            return false;
        }
        j++;
    }
    return true;
}
```

(b)

```
bool es_palindromo2(const vector<char> & s)
{
    int j = 0;
    while(j < s.size()/2)
    {
        if(s[j] != s[s.size()-j-1])
        {
            return false;
        }
        j++;
    }
    return true;
}
```

Ejercicio 13

(a)

```
void invertir_rango(vector<int> & v, const int & i, const int & j)
{
    // iteration var
    int k = i;
    while(k <= j)
    {
        int aux = v[k];
        v[k] = v[j];
        v[j] = aux;

        // inc
        k++;
    }
}
```

(b)

- $P_c : (0 \leq i < |v|) \land (0 \leq j < |v|) \land (i \leq j) \land (k = i)$
  i y j están en el rango de v. i es menor o igual que j. Para iterar, k es igual a i.
- $Q_c : (\forall k{:}\text{int})\,(0 \leq k < \min(i,j) \lor \max(i,j) < k < |v| \Rightarrow (v[k] = v_0[k]))$
  $\land\,(\min(i,j) \leq k \leq \max(i,j) \Rightarrow v[k] = v_0[\min(i,j) + \max(i,j) - k])$
  $\land (k = j + 1) \land (|v| = |v_0|)$
  Además de la postcondición del problema, también aclaré que $k = j + 1$ porque se tiene que negar la guarda, y se mantiene el tamaño del vector.
- $\mathcal{I} : (i \leq k \leq j + 1) \land (\forall k{:}\text{int})\,(0 \leq k < \min(i,j) \lor \max(i,j) < k < |v| \Rightarrow (v[k] = v_0[k]))$
  $\land\,(\min(i,j) \leq k \leq \max(i,j) \Rightarrow v[k] = v_0[\min(i,j) + \max(i,j) - k])$

Ejercicio 14

```
int contar_iguales(const vector<int> & v1, const vector<int> & v2)
Pre: |v1| = |v2|
Post: res = ∑_{k=0}^{|v1|-1} β(v1[k] = v2[k])
```

```
int res = 0;
int i = v1.size() - 1;
while (i >= 0){
    if(v1[i] == v2[i]){
        res = res + 1;
    }
    i = i - 1;
}
```

(a)

- $P_c : (|v_1| = |v_2|) \land (res = 0) \land (i = |v_1| - 1)$
  Agrego la precondición de la función y la declaración de las variables antes de que empiece el ciclo.
- $Q_c : \left(res = \sum_{i=0}^{|v_1|-1} \beta(v_1[k] = v_2[k])\right) \land (i = -1)$
  res será la sumatoria de todos los elementos que sean iguales en los dos vectores desde i hasta el final de un vector, ya que sabemos que los dos tienen el mismo tamaño.
- $\mathcal{I} : (-1 \leq i \leq |v_1| - 1) \land \left(res = \sum_{j=0}^{i+1} \beta(v_1[j] = v_2[j])\right)$
  i va desde $|v_1| - 1$ hasta -1 porque se tiene que negar la guarda.

(b)

```
vector<int> v1 = {0, (10), 2, (10), 3, (10)};
vector<int> v2 = {-1, (10), -2, (10), -3, (10)};
int i = 0;
int res = 2;
```

- Cumple el invariante, porque $-1 \leq 0 \leq 6 - 1$ y $2 = (0) + (1) + (0) + (1) + (0)$.
- Se cumple la guarda del ciclo, porque $0 \geq 0$.

(c)

En la última iteración sabíamos que $i = 0, res = 2$. Como se cumple la condición del *if*, $res = 2 + 1$. Finalmente, $i = -1$.

(d)

$res = 3, i = -1$.

- Esto cumple el invariante porque $3 = (1) + (0) + (1) + (0) + (1) + (0)$ y $-1 \leq -1 \leq 6$.
- No cumple con la guarda del ciclo porque $-1 \geq 0$ es falso.
- También cumple con la postcondición porque $i = -1$ y $3 = (1)+(0)+(1)+(0)+(1)+(0)$.

Ejercicio 15

(a)

```
void sumar_vector_a(vector<int> & v, vector<int> w, const int & k)
{
    // pre-conditions
    int i = 0;
    int res;

    // (0 <= i <= |v|)
    while(i < v.size())
    {
        // beta(v[i] > k)
        if(v[i] > k)
        {
            res += v[i];
        }
        i++;
    }
}
```

(b)

```cpp
void sumar_vector_b(vector<int> & v, vector<int> w)
{
    // pre-conditions
    int i = v.size();
    bool res = false;

    // (0 <= i <= |v|)
    while (i > 0)
    {
        if(v[i] % 2 == 0)
        {
            res = true;
            break;
        }
        i--;
    }
}
```

(c)

```cpp
void sumar_vector_c(vector<int> & v, vector<int> w)
{
    int i = 0;
    while (i < v.size())
    {
        if (v[i] = v[i+1])
        {
            continue;
        } else {
            break;
        }
        i++;
    }
}
```

(d)

```cpp
void sumar_vector_d(vector<int> & v, vector<int> w)
{
    int i = 0;
    while (i < v.size())
    {
        if(v[i] != w[i])
        {
            break;
        }
        i++;
    }

}
```