Departamento de Informática
Universidad de Valladolid
Campus de Segovia

TEMA 8: VERIFICACIÓN FORMAL DE ALGORITMOS

VERIFICACIÓN FORMAL DE ALGORITMOS

- Introducción. Conceptos preliminares.
- Concepto de corrección
- Reglas generales sobre precondiciones y postcondiciones
- Verificación de códigos sin bucles
- Bucles
- Terminación del programa

INTRODUCCIÓN
definiciones

- Definición:
  - La verificación formal de programas consiste en un conjunto de técnicas de comprobación formales que permiten demostrar si un programa funciona correctamente.

    - **Programa que funciona correctamente:** cumple con unas especificaciones dadas.
    - **Técnicas de comprobación:** La verificación consiste en un proceso de inferencia. Por tanto cada tipo de sentencia ejecutable posee una regla de inferencia.
    - **Representación formal:** Ternas de Hoare

INTRODUCCIÓN
Representación formal, Ternas de Hoare

- Si C es una parte del código, entonces cualquier aserción {P} se denomina precondición de C si {P} sólo implica al estado inicial. Cualquier aserción {Q} se denomina postcondición si {Q} sólo implica el estado final.
- Esta definición se representa como: {P}C{Q} y se denomina terna de Hoare.
- Ejemplo 1: $\{y \neq 0\}\ x:=1/y\ \{x=1/y\}$
- Ejemplo 2: A veces no existe precondición:
  - { } a:=b {a=b}
  - Esto significa que siempre se obtienen estados que satisfacen la postcondición independientemente de la precondición.
  - { } representa la aserción vacía que se puede entender como “verdadero para todos los estados”.

INTRODUCCIÓN
Conceptos preliminares

- Por código o parte de un código entendemos desde una sentencia o conjunto de sentencias hasta un programa.
- Como ya hemos comentado la verificación la realizaremos sobre programas escritos con un subconjunto de instrucciones de PASCAL :
  - Todos los operadores aritméticos y todas las funciones de la biblioteca de Pascal.
  - Bloques {Begin....End).
  - Construcciones del tipo if -then y if-then-else
  - Construcciones while.
- No se utilizarán declaraciones de tipo
- Ni sentencias de entrada/salida.
- La concatenación de dos códigos C_1 y C_2 supone la ejecución secuencial de dichos códigos y vendrá representada de la siguiente forma: $C_1;C_2$.

INTRODUCCIÓN
Conceptos preliminares

- Las aserciones o asertos son sentencias lógicas que hacen referencia a un estado del sistema. Para indicar que una sentencia es un aserto se encierra dicha sentencia entre corchetes {A}
- Un estado vendrá dado por el conjunto de valores que toman en ese instante el conjunto de variables que conforman la estructura de datos del problema.
- Precondiciones y postcondiciones:
  - Las **precondiciones** indican las condiciones que deben satisfacer los datos de entrada para que el programa pueda cumplir su tarea.
  - Las **postcondiciones** indican las condiciones de salida que son aceptables como soluciones correctas del problema en cuestión.
  - Ejemplo: Programa que calcula la raíz cuadrada
    - Precondición: Que el argumento de la función no sea negativo
    - Postcondición: La raíz cuadrada de ese argumento.

INTRODUCCIÓN
Conceptos preliminares

- Un programa es una secuencia de sentencias que transforman el estado inicial en un estado final.
- El estado inicial es el estado anterior a la ejecución de un código.
- El estado final es el estado posterior a la ejecución de dicho código.
- El estado del sistema viene determinado por los valores de las variables en cada momento.

INTRODUCCIÓN
Nomenclatura

- En la mayoría de los códigos el estado final depende del estado inicial. Por tanto Precondición y Postcondición no son independientes entre sí.

- Hay dos posibles nomenclaturas que permitan reflejar dicha dependencia:
  - **Representación del súbíndice**: mediante subíndices se indica si las variables representan valores iniciales o finales. $\{a_\omega=b_\alpha\} \wedge \{b_\omega=a_\alpha\}$ donde 'ω' repesenta el estado final y 'α' el estado inicial.
  - **Representación de las variables ocultas**: Las variables ocultas son variables que aparecen o no en el código y que se introducen para almacenar los valores iniciales de ciertas posiciones de memoria. Su definición debe aparecer en la precondición.

    {a=A,b=B} h:=a; a:=b; b:=h {a=B, b=A}

CONCEPTO DE CORRECCIÓN

- Si {P}C{Q} es un código con la precondición {P} y la postcondición {Q}, entonces {P}C{Q} es correcto si cada estado inicial posible que satisfaga {P} da como resultado un estado final que satisface {Q}.
- Dentro de este concepto debemos distinguir entre corrección total y parcial.
  - **Corrección parcial**: se dice que {P}C{Q} es parcialmente correcto si el estado final de C, cuando termina el programa (aunque no se le exige esta premisa), satisface {Q} siempre que el estado inicial satisface {P}.
  - **Corrección total**: Se da cuando un código además de ser correcto parcialmente termina.
- Los códigos sin bucles siempre terminan por lo que la corrección parcial implica la corrección total. Esta distinción es esencial sólo en el caso de códigos que incluyan bucles o recursiones.

REGLAS GENERALES RELATIVAS A LAS PRECONDICIONES Y POSTCONDICIONES

- Si R y S son dos aserciones entonces se dice que R es más fuerte que S si R$\Rightarrow$S. Si R es más fuerte que S entonces se dice que S es más débil que R.
  - **Ejemplo:** i>1 es más fuerte que i>0 ya que siempre que i>1 esto implica que i>0. Por tanto i>1 $\Rightarrow$i>0.
- Si una aserción R es más fuerte que una aserción S entonces todos los estados que satisfacen R también satisfacen S pero no viceversa. Un fortalecimiento de la aserción disminuye el número de estados que la pueden satisfacer.
  - **Más fuerte** $\Rightarrow$ más selectivo, más específico.
  - **Más débil** $\Rightarrow$ menos selectivo, más general.
  - La aserción más débil es { } que recoge todos los estados posibles. Constante lógica V (true).
  - La aserción más fuerte será por tanto F (false), ya que representa que ningún estado satisface dicha condición.

FORTALECIMIENTO DE LAS PRECONDICIONES

- Si una parte de un código es correcta bajo la precondición {P}, entonces permanecerá correcto si se refuerza {P}.
- Si {P}C{Q} es correcto y además $P_1 \Rightarrow P$ entonces se puede afirmar que $\{P_1\}C\{Q\}$ es correcto. Esto conduce a la siguiente regla de inferencia:

$$\frac{P_1 \Rightarrow P \qquad \{P\}C\{Q\}}{\{P_1\}C\{Q\}}$$

  - Ejemplo: supongamos que la siguiente terna de Hoare es correcta: $\{y \neq 0\}$ x:=1/y $\{x=1/y\}$, demostrar que también lo es $\{y=4\}$ x:=1/y $\{x=1/y\}$.

$$\frac{y=4 \Rightarrow y \neq 0 \qquad \{y \neq 0\} \text{ x:=1/y } \{x=1/y\}}{\{y=4\} \text{ x:=1/y } \{x=1/y\}}$$

FORTALECIMIENTO DE LA PRECONDICIÓN VACÍA

- La aserción vacía puede ser reforzada para producir cualquier precondición {P} sea cual fuere.

- Ejemplo:
  - { } a:=b {a=b} se puede utilizar para justificar {P} a:=b {a=b} donde {P} puede ser cualquier condición imaginable.

- Como regla general:
  - siempre será ventajoso intentar formular la precondición más débil posible que asegure que se cumple una cierta postcondición dada. De esta forma cualquier precondición que implique la anterior será satisfecha automáticamente.
  - Además los programas deben ser escritos de modo que resulten lo más versátiles posibles (Generalidad).

DEBILITAMIENTO DE POSTCONDICIONES

- El principio de debilitamiento de postcondiciones permite concluir que $\{P\}C\{Q_1\}$ una vez que $\{P\}C\{Q\}$ y $Q \Rightarrow Q_1$ hayan sido establecidos formalmente. Esto conduce a la siguiente regla de inferencia:

$$\frac{\{P\}C\{Q\} \qquad Q \Rightarrow Q_1}{\{P\}C\{Q_1\}}$$

- Ejemplo: Si $\{\ \}$ max:=b $\{max=b\}$ es correcto, demostrar que se cumple: $\{\ \}$ max:=b $\{max \geq b\}$

$$\frac{\{\ \} \text{ max:=b } \{max=b\} \qquad max=b \Rightarrow max \geq b}{\{\ \} \text{ max:=b } \{max \geq b\}}$$

REGLA DE LA CONJUNCIÓN

- La siguiente regla permite reforzar la precondición y postcondición de forma simultánea.

- Definición: Si C es una parte de un código y se ha establecido que $\{P_1\}C\{Q_1\}$ y $\{P_2\}C\{Q_2\}$ se puede afirmar que $\{P_1 \wedge P_2\}C\{Q_1 \wedge Q_2\}$. Formalmente esto se expresa como:

$$\{P_1\}C\{Q_1\}$$
$$\underline{\{P_2\}C\{Q_2\}}$$
$$\{P_1 \wedge P_2\}C\{Q_1 \wedge Q_2\}$$

- Como caso particular:

$$\{\ \}C\{Q_1\}$$
$$\underline{\{P_2\}C\{Q_2\}}$$
$$\{P_2\}C\{Q_1 \wedge Q_2\}$$

REGLA DE LA CONJUNCIÓN

- Ejemplo:
  - Aplicar las ternas $\{\ \}i:=i+1\ \{i_\omega:=i_\alpha+1\}$, $\{i_\alpha>0\ \}i:=i+1\ \{i_\alpha>0\}$, para demostrar que: $\{i>0\ \}i:=i+1\ \{i>1\}$

$$\{\ \}i:=i+1\{i_\omega:=i_\alpha+1\}$$
$$\underline{\{i_\alpha>0\ \}i:=i+1\ \{i_\alpha>0\}}$$
$$\{i_\alpha>0\ \}i:=i+1\ \{(i_\alpha>0)\wedge(i_\omega:=i_\alpha+1)\}$$

  - De donde: $i_\alpha:=i_\omega-1$ y puesto que $i_\alpha>0$ entonces $i_\omega>1$ por tanto:

$$\{i>0\ \}i:=i+1\ \{i>1\}$$

REGLA DE LA DISYUNCIÓN

- La siguiente regla permite debilitar la precondición y postcondición de forma simultánea.

- Definición: Si C es una parte de un código y se ha establecido que $\{P_1\}C\{Q_1\}$ y $\{P_2\}C\{Q_2\}$ se puede afirmar que $\{P_1 \vee P_2\}C\{Q_1 \vee Q_2\}$. Formalmente esto se expresa como:

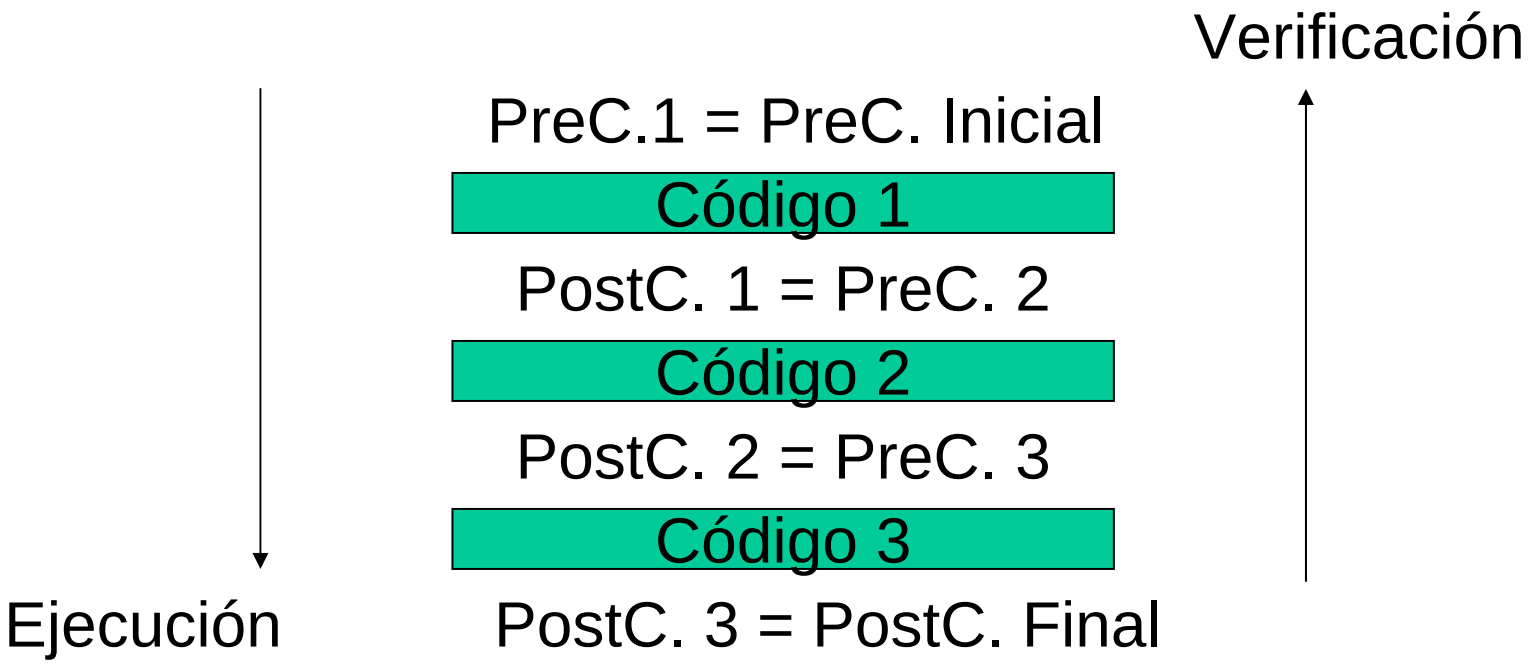
$$\{P_1\}C\{Q_1\}$$
$$\underline{\{P_2\}C\{Q_2\}}$$
$$\{P_1 \vee P_2\}C\{Q_1 \vee Q_2\}$$

- Como caso particular:

$$\{\ \}C\{Q_1\}$$
$$\underline{\{P_2\}C\{Q_2\}}$$
$$\{P_2\}C\{Q_1 \vee Q_2\}$$

VERIFICACIÓN DE CÓDIGOS SIN BUCLES

- **Verificación:** Para demostrar la corrección de las partes de un programa se parte de la postcondición final del código, es decir de las condiciones que deben satisfacer los resultados. A partir de ahí se deduce la precondición.
- El código se verifica en sentido contrario a como se ejecuta.



SENTENCIAS DE ASIGNACIÓN

- La regla de asignación requiere que las variables implicadas no compartan el mismo espacio de memoria. Esto excluye el uso de punteros y de variables con índices (arrays).

- Las sentencias de asignación son sentencias de la forma V:=E, en donde V es una variable y E es una expresión.

$$\{\ \}V:=E\{\ V=E\ _{\alpha}\}$$

- Sin embargo esta expresión no siempre es correcta (x:=1/y). En dicho caso debemos generalizar:

REGLA DE LA ASIGNACIÓN

- **Regla de la asignación:** Si C es una sentencia de la forma V:=E con la postcondición {Q}, entonces la precondición de C puede hallarse sustituyendo todos los caso de V en Q por E. Si Q_E^V es la expresión que se obtiene mediante esto, entonces:

$$\{P\}V:=E\{Q\}$$

$$\{P\}=\{Q_E^V\} \Rightarrow \{ V=E_{\alpha},Q\}$$

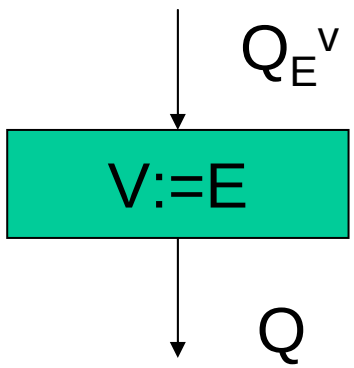
$$\{Q_E^V\}V:=E\{Q\}$$

- Ejemplo: Determinar la precondición para que la terna siguiente sea correcta: {P} i:=2*i {i<6}

$$\{Q_E^V\}=\{i_{\omega}=2*i_{\alpha},\ i_{\omega}<6\}\Rightarrow\{2*i<6\} \Rightarrow\{i<3\}$$

SENTENCIAS DE ASIGNACIÓN

$$\{Q_E^V\}V:=E\{Q\}$$



SENTENCIAS DE ASIGNACIÓN
Ejemplos

- **Ejemplo 1**: Determinar la precondición para que la terna siguiente sea correcta: {P} j:=i+1 {j>0}

  $\{Q_E^V\}=\{j_\omega=i_\alpha+1,\ j_\omega>0\}\Rightarrow\{i_\alpha+1>0\}$

- **Ejemplo 2**: Determinar la precondición para que la terna siguiente sea correcta: {P} $y:=x^2$ {y>1}

  $\{Q_E^V\}=\{y_\omega=x_\alpha * x_\alpha,\ y_\omega>1\}\Rightarrow\{x_\alpha^2>1\}$

- **Ejemplo 3**: Determinar la postcondición para que la terna siguiente sea correcta: {x>2} $x:=x^2$ {Q}

  $\{Q\}\Rightarrow\{x_\alpha>2,\ x_\omega=x_\alpha^2\}\Rightarrow\{x_\omega>4\}$

- **Ejemplo 4**: Determinar la precondición para que la terna siguiente sea correcta: {P} x:=1/x $\{x\geq 0\}$

  $\{Q_E^V\}\Rightarrow\{x_\omega=1/x_\alpha,\ x_\omega\geq 0\}\Rightarrow\{x_\alpha>0\}$

CONCATENACIÓN DE CÓDIGO

- La concatenación significa que las partes del programa se ejecutan secuencialmente de tal forma que el estado final de la primera parte de un código se convierte en el estado inicial de la segunda parte del programa.

- **Regla de la concatenación:** Sean C_1 y C_2 dos partes de un código y sea $C_1;C_2$ su concatenación. Si $\{P\}C_1\{R\}$ y $\{R\}C_2\{Q\}$ Son ternas de Hoare correctas entonces se puede afirmar que:

$$\{P\}C_1\{R\}$$
$$\underline{\{R\}C_2\{Q\}}$$
$$\{P\}C_1;C_2\{Q\}$$

CONCATENACIÓN DE CÓDIGO
Ejemplo 1

- **Ejemplo:** Demostrar que el siguiente código es correcto:

$$\{\ \}c:=a+b;\ c:=c/2\ \{c=(a+b)/2\}$$

$$\{\ P\}c:=c/2\ \{c=(a+b)/2\}$$

$$P=\{c_{\omega}=c/2, c_{\omega}=(a+b)/2\}\Rightarrow\{c/2=(a+b)/2\}$$

$$\{\ c/2=(a+b)/2\}c:=c/2\ \{c=(a+b)/2\}$$

$$\{P\}\ c:=a+b\ \{c/2=(a+b)/2\}$$

$$P=\{c_{\omega}=a+b, c_{\omega}/2=(a+b)/2\}\Rightarrow\{(a+b)/2=(a+b)/2\}\Rightarrow\{\ \}$$

$$\{\ \}c:=a+b\ \{c/2=(a+b)/2\}$$

Con lo que queda demostrado.

EJEMPLO 2

- **Ejemplo:** Demostrar que el siguiente código es correcto: { }s:=1; s:=s+r; s=s+r*r {s=1+r+r^2}

$$\{ P\}s:=s+r*r \{s=1+r+r^2\}$$

$$P=\{s_\omega=s+r^2, s_\omega=1+r+r^2\}\Rightarrow\{s+r^2=1+r+r^2\}$$

$$\{ s+r^2=1+r+r^2\}s:=s+r*r \{s=1+r+r^2\}$$

$$\{P\}\ s:=s+r \{s=1+r\}$$

$$P=\{s_\omega=s+r, s_\omega=1+r\}\Rightarrow\{s+r=1+r\}$$

$$\{s+r=1+r \}s:=s+r \{s=1+r\}$$

$$\{P\}\ s:=1 \{s=1\}$$

$$P=\{s_\omega=1, s_\omega=1\}\Rightarrow\{1=1\} \Rightarrow\{ \}$$

$$\{ \}s:=1 \{s=1\}$$

Con lo que queda demostrado.

EJEMPLO 3

- **Ejemplo:** Demostrar que el siguiente código es correcto: {a=A,b=B}h:=a; a:=b; b=h {a=B,b=A}

$$\{ P\}b:=h \{a=B,b=A\}$$

$$P=\{b=h, a=B, b=A\}\Rightarrow\{h=A, a=B\}$$

$$\{ h=A, a=B\}b:=h \{a=B, b=A\}$$

$$\{P\}\ a:=b \{h=A, a=B\}$$

$$P=\{a=b, h=A, a=B\}\Rightarrow\{h=A, b=B\}$$

$$\{h=A, b=B\}a:=b \{h=A, a=B\}$$

$$\{P\}\ h:=a \{h=A, b=B\}$$

$$P=\{h=a, h=A,b=B\}\Rightarrow\{a=A, b=B\}$$

$$\{a=A, b=B\}h:=a \{h=A, b=B\}$$

Con lo que queda demostrado.

INVARIANTE DE UN CÓDIGO

- Cualquier aserción que es a la vez precondición y postcondición de una parte del código se denomina invariante.

  - **Ejemplo:** Demostrar que $r=2^i$ es un invariante del siguiente código: i:=i+1; r:=r*2.

$$\{ P\}r:=r*2 \{r=2^i\}$$

$$P=\{r=r*2, r=2^i\}\Rightarrow\{r*2=2^i\}\Rightarrow\{r=2^{i-1}\}$$

$$\{r=2^{i-1}\}\ r:=r*2\ \{r=2^i\}$$

$$\{P\}\ i:=i+1\ \{r=2^{i-1}\}$$

$$P=\{i=i+1, r=2^{i-1}\}\Rightarrow\{r=2^{i+1-1}\}\Rightarrow\{r=2^i\}$$

$$\{r=2^i\}i:=i+1\ \{r=2^{i-1}\}$$

Con lo que queda demostrado.

LA SENTENCIA IF
con clausula else

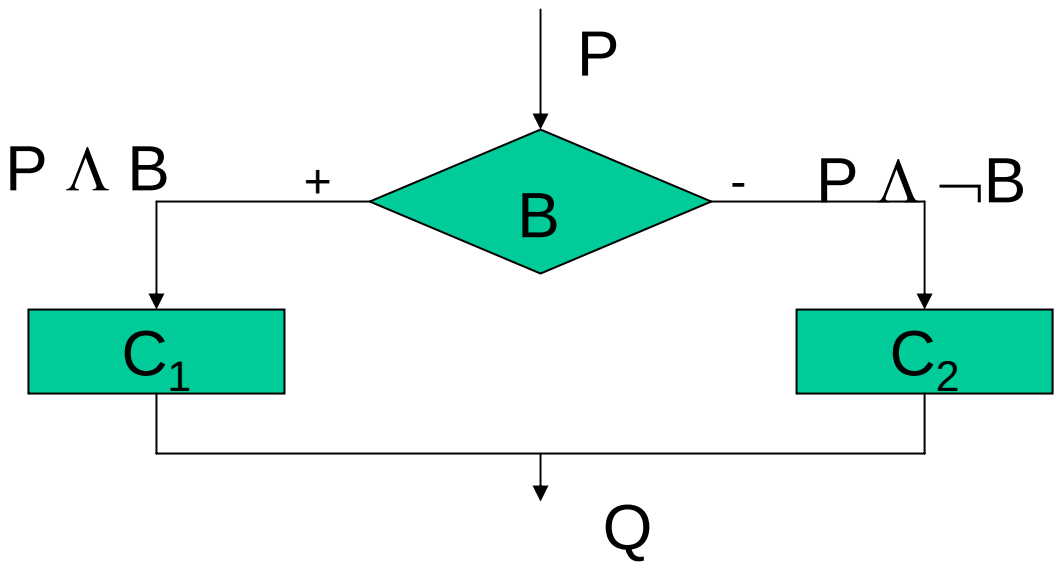
- Si C_1 y C_2 son dos partes de un programa y si B es una condición, entonces la sentencia '**if B then C_1 else C_2**' se interpreta de la siguiente forma: si B es verdadero se ejecuta C_1 sino C_2.
- Si ahora se desea demostrar la corrección de una sentencia 'if' con una precondición {P} y una postcondición {Q} tendremos dos posibilidades:
  - Si el estado inicial satisface B además de P entonces se ejecutará C_1 y por tanto la verificación equivaldrá a demostrar que $\{P \wedge B\}\ C_1\ \{Q\}$ es correcto.
  - Si el estado inicial no satisface B entonces se ejecutará C_2 y por tanto la verificación equivaldrá a demostrar que $\{P \wedge \neg B\}\ C_2\ \{Q\}$ es correcto.

REGLA PARA CONDICIONES CON CLAUSULA ELSE

$$\{P \wedge B\} C_1 \{Q\}$$

$$\underline{\{P \wedge \neg B\} C_2 \{Q\}}$$

$$\{P\} \text{if } B \text{ then } C_1 \text{ else } C_2 \{Q\}$$



LA SENTENCIA IF
con clausula else, Ejemplo

- **Ejemplo:** Demostrar que:

  $\{\ \}$if a>b then m:=a else m:=b$\{(m \geq a) \wedge (m \geq b)\}$

- $\{\ a>b\}m:=a\ \{(m \geq a) \wedge (m \geq b)\}$

$$P=\{m=a,\ (m \geq a) \wedge (m \geq b)\} \Rightarrow \{a \geq a,\ a \geq b\}$$

$$\{a=a\} \Rightarrow \{a \geq a\},\ \{a=a,\ a \geq b\} \Rightarrow \{a \geq b\}$$

$$\{\ a>b\} \Rightarrow \{a \geq b\}\ \{\text{Fortalecimiento PreC.}\}$$

$$\{\ a>b\}m:=a\ \{(m \geq a) \wedge (m \geq b)\}$$

- $\{\ \neg(a>b)\}m:=b\ \{(m \geq a) \wedge (m \geq b)\}$

$$P=\{m=b,(m \geq a) \wedge (m \geq b)\} \Rightarrow \{b \geq a,\ b \geq b\}$$

$$\{b=b\} \Rightarrow \{b \geq b\},\ \{b \geq a,\ b=b\} \Rightarrow \{b \geq a\}$$

$$\{\ \neg(a>b)\} \Rightarrow \{b \geq a\}$$

$$\{\ b \geq a\}m:=b\ \{(m \geq a) \wedge (m \geq b)\}$$

Con lo que queda demostrado.

LA SENTENCIA IF
sin clausula else

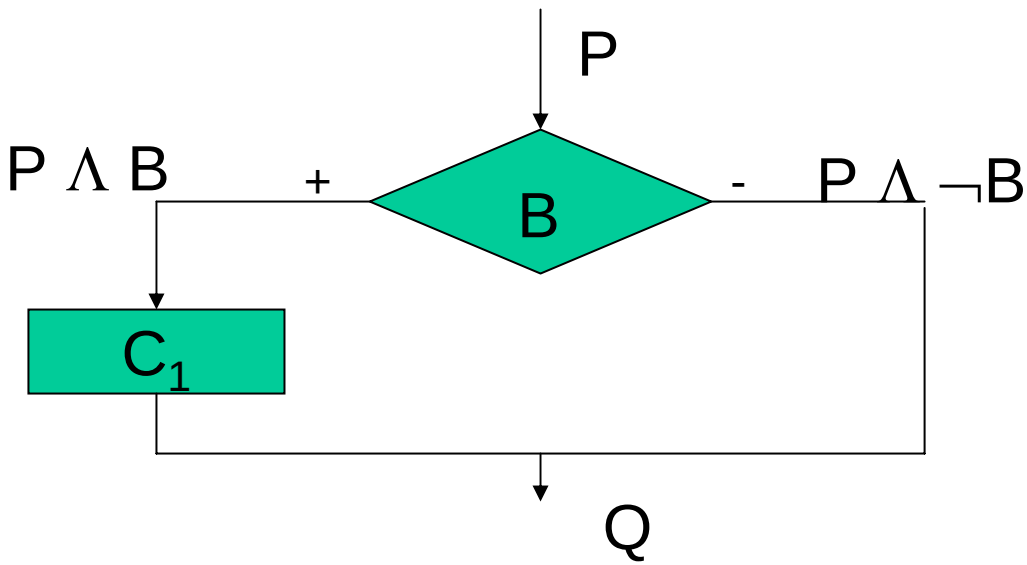
- Si C_1 es una parte de un programa y B es una condición, entonces la sentencia '**if B then C_1** ' se interpreta de la siguiente forma: si B es verdadero se ejecuta C_1.
- Si ahora se desea demostrar la corrección de una sentencia 'if' con una precondición {P} y una postcondición {Q} tendremos dos posibilidades:
  - Si el estado inicial satisface B además de P entonces se ejecutará C_1 y por tanto la verificación equivaldrá a demostrar que $\{P \wedge B\}\ C_1\ \{Q\}$ es correcto.
  - Si el estado inicial no satisface B entonces esto equivaldrá a demostrar que $\{P \wedge \neg B\} \Rightarrow \{Q\}$.

REGLA PARA CONDICIONES SIN CLAUSULA ELSE

$$\{P \wedge B\} C_1 \{Q\}$$

$$\underline{\{P \wedge \neg B\} \Rightarrow \{Q\}}$$

$$\{P\} \text{if } B \text{ then } C_1 \{Q\}$$



LA SENTENCIA IF
sin clausula else, Ejemplo

- **Ejemplo:** Demostrar que:
  - { }if max<a then max:=a {(max≥a)}

- {¬(max<a)}⇒{(max≥a)}

  Lo que es obvio

- { max<a} max:=a {(max≥a)}

  P={max=a, (max≥a)}⇒{a≥a}

  {a≥a} ⇒{ },

  Y puesto que {max<a} ⇒ { }

Con lo que queda demostrado.

REGLA PARA DETERMINAR LA PREC. A PARTIR DE LA POSTC. EN ESTRUCTURAS IF.

- Cuando se trabaja con la estructura ‘if’ es bastante fácil obtener la postcondición a partir de la precondición. Sin embargo normalmente lo que se necesita es lo contrario.

- Para facilitar la labor anterior es mejor utilizar la siguiente regla de inferencia:

$$\{B\}C_1\{B\}$$

$$\{\neg B\}C_2\{\neg B\}$$

$$\{P\}\ C_1\{Q_1\}$$

$$\underline{\{P\}\ C_2\{Q_2\}}$$

$$\{P\}\text{if B then } C_1 \text{ else } C_2\{(B\wedge Q_1)\vee (\neg\ B\wedge Q_2)\}$$

BUCLES

- Mientras que todos los programas sin bucles terminan, no se puede decir lo mismo de aquellos que si lo poseen o son recursivos. Por ahora sólo nos preocupará la corrección parcial.
- El **invariante del bucle** es un aserto que captura las características que no varían al ejecutarse un bucle.
- El **variante de un bucle** es una expresión que mide el progreso realizado por el bucle hasta satisfacer la condición de salida.
- Las estructuras iterativas elegidas para establecer su corrección son:
    - 'while B do C', donde B es la condición de entrada y C es el código contenido dentro del bucle.
    - 'repeat C until B', donde B es la condición de salida y C es el código contenido dentro del bucle.
- Como podremos comprobar en cada iteración el bucle debe progresar hacia la postcondición final, que debe derivarse del invariante cuando la condición de entrada no se cumple (condición de salida).

ALGUNOS INDICIOS PARA ENCONTRAR EL INVARIANTE

- El invariante del bucle es el precursor de la postcondición, lo que indica que este debe ser parecido a la postcondición.
- En cada iteración debe progresar hacia la postcondición, por lo que deberá contener variables que son modificadas dentro del bucle.
- En última instancia el invariante es solamente una formalización de los objetivos del programador.

EJEMPLO I PARA LA DETERMINACIÓN DE UN INVARIANTE

- Encontrar el invariante asociado al código interno del bucle:

```
sum:=0;
j:=0;
while  j<>n do begin
            sum:=sum+a;
            j:=j+1
end:
```

EJEMPLO I

- Si el invariante es **I={sum=j*a}**

$$\{ P\}\ j:=j+1;\ \{sum=j*a\}$$

$$P=\{j=j+1, sum=j*a\} \Rightarrow \{sum=(j+1)*a\}$$

$$\mathbf{\{sum=(j+1)*a\}\ j:=j+1\ \{sum=j*a\}}$$

$$\{P\}\ sum:=sum+a;\ \{sum=(j+1)*a\}$$

$$P=\{sum=sum+a,\ sum=(j+1)*a\} \Rightarrow \{sum+a=(j+1)*a\}$$

$$\mathbf{\{sum=j*a\}\ sum:=sum+a;\ \{sum=(j+1)*a\}}$$

Con lo que queda demostrado que {I}C{I}

$$\{P\}\ j:=0\ \{sum=j*a\}$$

$$P=\{j=0, sum=j*a\} \Rightarrow \{sum=0\}$$

$$\mathbf{\{sum=0\}\ j:=0\ \{sum=j*a\}}$$

$$\{P\}\ sum:=0\ \{sum=0\}$$

$$P=\{sum=0, sum=0\} \Rightarrow \{\ \}$$

$$\mathbf{\{\ \}\ sum:=0\ \{sum=0\}}$$

$$\{(\neg\ B \wedge I)\} \Rightarrow \{(sum=j*a) \wedge (j=n)\} \Rightarrow \{sum=n*a\}$$

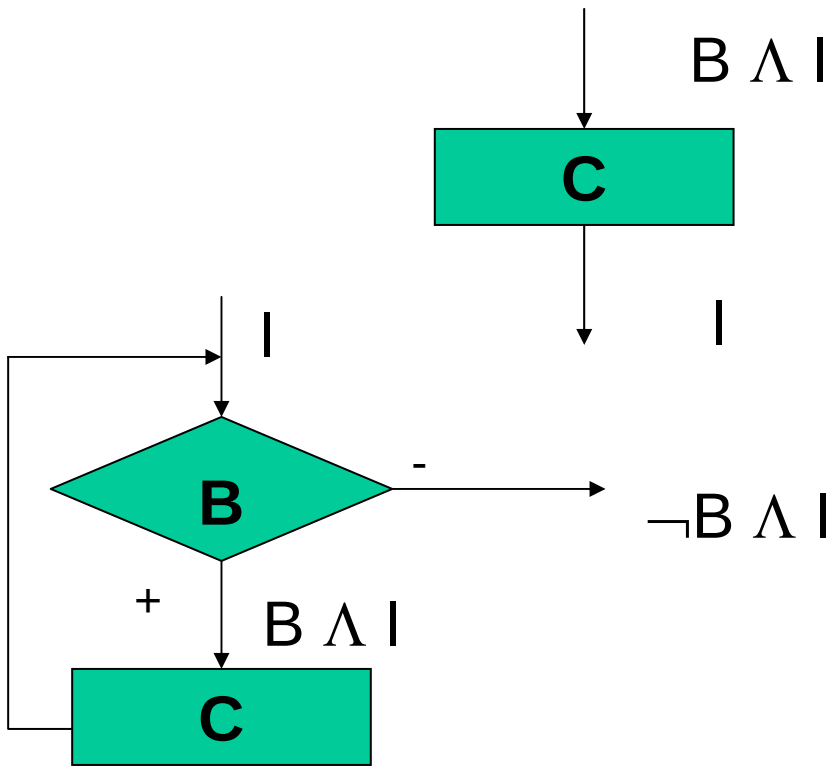
REGLA DE INFERENCIA PARA EL BUCLE WHILE.

- Si C es un código tal que se cumple $\{B \wedge I\}\ C\ \{I\}$ entonces se puede inferir lo siguiente:

$$\frac{\{B \wedge I\}\ C\ \{I\}}{\{I\}\ \text{while } B \text{ do } C\ \{(\neg B \wedge I)\}}$$

ENUNCIADO REPETITIVO TIPO WHILE

$$\frac{\{B \wedge I\}\ C\ \{I\}}{\{I\}\ \text{while } B \text{ do } C\ \{(\neg B \wedge I)\}}$$



EJEMPLO I PARA LA DETERMINACIÓN DE LA CORRECCIÓN (while)

- Determinar la corrección del siguiente código:

```
i:=1;
sum:=0;
while not ( i=n+1) do begin
        sum:=sum+i*i;
        i:=i+1
end:
```

- Se supone que este código halla la suma de los cuadrados de 1 a n.

EJEMPLO I

- Si la postcondición es ={(sum=$\Sigma_{j=1}^{n} j^2$) $\wedge$ (i=n+1)} y el invariante:

I= {(sum=$\Sigma_{j=1}^{i-1} j^2$) $\wedge$ (i $\leq$ n+1)}

{ P} i:=i+1; {(sum=$\Sigma_{j=1}^{i-1} j^2$) $\wedge$ (i $\leq$ n+1)}

P={i=i+1,sum= $\Sigma_{j=1}^{i-1} j^2$, i $\leq$ n+1}$\Rightarrow${sum= $\Sigma_{j=1}^{i} j^2$, i+1 $\leq$ n+1}

{ i+1 $\leq$ n+1} $\Rightarrow$ {i $\leq$ n} $\Rightarrow${i<n+1}

{(sum=$\Sigma_{j=1}^{i} j^2$) $\wedge$ (i <n+1)} i:=i+1 {(sum=$\Sigma_{j=1}^{i-1} j^2$) $\wedge$ (i $\leq$ n+1)}

{P} sum:=sum+i^2; {(sum=$\Sigma_{j=1}^{i} j^2$) $\wedge$ (i < n+1)}

P={sum=sum+i^2, sum= $\Sigma_{j=1}^{i} j^2$, i < n+1} $\Rightarrow$ {sum+i^2= $\Sigma_{j=1}^{i} j^2$, i<n+1}

{sum+i^2= i^2+$\Sigma_{j=1}^{i-1} j^2$, i<n+1} $\Rightarrow$ {sum= $\Sigma_{j=1}^{i-1} j^2$, i< n+1}

{(sum=$\Sigma_{j=1}^{i-1} j^2$) $\wedge$ (i<n+1)} sum:=sum+i^2; {(sum=$\Sigma_{j=1}^{i} j^2$) $\wedge$ (i < n+1)}

Con lo que queda demostrado que {B$\wedge$ I}C{I}

{P} sum:=0 {(sum=$\Sigma_{j=1}^{i-1} j^2$) $\wedge$ (i$\leq$n+1)}

P={sum=0, sum= $\Sigma_{j=1}^{i-1} j^2$, i $\leq$ n+1}$\Rightarrow${0= $\Sigma_{j=1}^{i-1} j^2$, i $\leq$ n+1}

{(0= $\Sigma_{j=1}^{i-1} j^2$) $\wedge$ (i $\leq$ n+1)} sum:=0 {(sum= $\Sigma_{j=1}^{i-1} j^2$) $\wedge$ (i $\leq$ n+1)}

{P} i:=1 {(0= $\Sigma_{j=1}^{i-1} j^2$) $\wedge$ (i $\leq$ n+1)}

P={i=1, 0= $\Sigma_{j=1}^{i-1} j^2$, i<n+1}$\Rightarrow${0= $\Sigma_{j=1}^{0} j^2$, 1 $\leq$ n+1}$\Rightarrow${0 $\leq$ n}

{n $\geq$0 } i:=1 {(0= $\Sigma_{j=1}^{i-1} j^2$) $\wedge$ (i<n+1)}

{($\neg$ B$\wedge$I)} $\Rightarrow$ {(i=n+1) $\wedge$ ((sum=$\Sigma_{j=1}^{i-1} j^2$) $\wedge$ (i $\leq$ n+1))} $\Rightarrow$

{(sum=$\Sigma_{j=1}^{n} j^2$) $\wedge$ (i=n+1)}

EJEMPLO I

- Si la postcondición es $\{(sum=\Sigma_{j=1}^{n} j^2) \wedge (i=n+1)\}$ y el invariante:

$$\mathbf{I= \{(sum=\Sigma_{j=1}^{i-1} j^2) \wedge (i \leq n+1)\}}$$

$$\mathbf{\{n \geq 0\}}\ i:=1\ \{(0= \Sigma_{j=1}^{i-1} j^2) \wedge (i<n+1)\}$$

$$\{(0= \Sigma_{j=1}^{i-1} j^2) \wedge (i \leq n+1)\}\ sum:=0\ \mathbf{\{(sum= \Sigma_{j=1}^{i-1} j^2) \wedge (i \leq n+1)\}}$$

while not (i=n+1) do

$$\mathbf{\{(sum=\Sigma_{j=1}^{i-1} j^2) \wedge (i<n+1)\}}\ sum:=sum+i^2;\ \{(sum=\Sigma_{j=1}^{i} j^2) \wedge (i <n+1)\}$$

$$\{(sum=\Sigma_{j=1}^{i} j^2) \wedge (i <n+1)\}\ \ i:=i+1\ \ \mathbf{\{(sum=\Sigma_{j=1}^{i-1} j^2) \wedge (i \leq n+1)\}}$$

$$\frac{\{B \wedge I\}\ C\ \{I\}}{\{I\}\ \text{while } B \text{ do } C\ \{(\neg B \wedge I)\}}$$

$$\{(\neg B \wedge I)\} \Rightarrow \{(i=n+1) \wedge ((sum=\Sigma_{j=1}^{i-1} j^2) \wedge (i \leq n+1))\} \Rightarrow$$

$$\{(sum=\Sigma_{j=1}^{n} j^2) \wedge (i=n+1)\}$$

EJEMPLO II PARA LA DETERMINACIÓN DE LA CORRECCIÓN (while)

- Determinar la corrección del siguiente código:

```
f:=1;
t:=0;
i:=0;
while  ( i ≤ n) do begin
        t:=t+f;
        f:=f*r;
        i:=i+1
end;
```

- Se supone que este código calcula:

$$t=\Sigma_{j=0}^{n} r^j \text{ cuando } i>n$$

EJEMPLO II

- Si la postcondición es $\{(t=\Sigma_{j=0}^{n} r^j) \wedge (i>n)\}$ y el invariante:

$$\mathbf{I= \{(t=\Sigma_{j=0}^{i-1} r^j)\wedge(f=r^i)\wedge(i\leq n+1)\}}$$

$$\{ P\}\ i:=i+1;\ \{(t=\Sigma_{j=0}^{i-1} r^j)\wedge(f=r^i)\wedge(i\leq n+1)\}$$

$$P=\{i=i+1, t=\Sigma_{j=1}^{i-1} r^j,\ f=r^i,\ i \leq n+1\} \Rightarrow \{t=\Sigma_{j=1}^{i} r^j,\ f=r^{i+1},\ i+1 \leq n+1\}$$

$$\{ i+1 \leq n+1\} \Rightarrow \{i \leq n\} \Rightarrow \{i<n+1\}$$

$$\mathbf{\{(t=\Sigma_{j=1}^{i} r^j)\wedge (f=r^{i+1})\wedge(i <n+1)\}\ i:=i+1\ \{(t=\Sigma_{j=1}^{i-1} r^j)\wedge(f=r^i)\wedge (i \leq n+1)\}}$$

$$\{P\}\ f:=f*r;\ \{(t=\Sigma_{j=1}^{i} r^j)\wedge (f=r^{i+1})\wedge(i <n+1)\}$$

$$P= \{f=f*r,\ t=\Sigma_{j=1}^{i} r^j,\ f=r^{i+1},\ i <n+1\} \Rightarrow \{t=\Sigma_{j=1}^{i} r^j,\ f*r=r^{i+1},\ i <n+1\} \Rightarrow$$

$$\Rightarrow \{t=\Sigma_{j=1}^{i} r^j,\ f=r^i,\ i <n+1\}$$

$$\mathbf{\{(t=\Sigma_{j=1}^{i} r^j)\wedge(f=r^i)\wedge(i <n+1)\}\ f:=f*r;\ \{(t=\Sigma_{j=1}^{i} r^j)\wedge (f=r^{i+1})\wedge(i <n+1)\}}$$

$$\{P\}\ t:=t+f;\ \{(t=\Sigma_{j=1}^{i} r^j)\wedge (f=r^i)\wedge(i <n+1)\}$$

$$P= \{t=t+f,\ t=\Sigma_{j=1}^{i} r^j,\ f=r^i,\ i <n+1\} \Rightarrow \{t+f=\Sigma_{j=1}^{i} r^j,\ f=r^i,\ i <n+1\} \Rightarrow$$

$$\Rightarrow \{t+r^i=r^i+\Sigma_{j=1}^{i-1} r^j,\ f=r^i,\ i <n+1\} \Rightarrow \{t=\Sigma_{j=1}^{i-1} r^j,\ f=r^i,\ i <n+1\}$$

$$\mathbf{\{(t=\Sigma_{j=1}^{i-1} r^j)\wedge(f=r^i)\wedge(i <n+1)\}\ f:=f*r;\ \{(t=\Sigma_{j=1}^{i} r^j)\wedge (f=r^i)\wedge(i <n+1)\}}$$

Con lo que queda demostrado que $\{B\wedge I\}C\{I\}$

EJEMPLO II

$$\{P\}\ i:=0;\ \{(t=\Sigma_{j=1}^{i-1} r^j)\wedge (f=r^i)\wedge(i \leq n+1)\}$$

$$P=\{i=0,\ t=\Sigma_{j=1}^{i-1} r^j,\ f=r^i,\ i\leq n+1\}\Rightarrow \{t=\Sigma_{j=1}^{-1} r^j,\ f=r^0,\ 0 \leq n+1\} \Rightarrow$$

$$\Rightarrow\{t=0,\ f=1,\ 0 \leq n+1\}$$

$$\mathbf{\{(t=0\ )\wedge(f=1)\wedge(0<n+1)\}\ i:=0;\ \{(t=\Sigma_{j=1}^{i-1} r^j)\wedge\ (f=r^i)\wedge(i \leq n+1)\}}$$

$$\{P\}\ t:=0;\ \{(t=0)\wedge\ (f=1)\wedge(-1\leq n)\}$$

$$P=\{t=0,\ t=0,\ f=1,\ -1 \leq n\}\Rightarrow \{0=0,\ f=1,\ -1 \leq n\}$$

$$\mathbf{\{(f=1)\wedge(-1\leq n)\}\ t:=0;\ \{(t=0)\wedge\ (f=1)\wedge(-1 \leq n)\}}$$

$$\{P\}\ f:=1;\ \ (f=1)\wedge(-1\leq n)\}$$

$$P=\{f=1,\ f=1,\ -1 \leq n\}\Rightarrow \{1=1,\ -1 \leq n\}$$

$$\mathbf{\{(-1\leq n)\}\ f:=1;\ \{(f=1)\wedge(-1 \leq n)\}}$$

$$\{(i>n)\} \Rightarrow\{i=n+1\}$$

$$\{(\neg\ B\wedge I)\} \Rightarrow \{(i=n+1)\wedge((t=\Sigma_{j=0}^{i-1} r^j)\wedge(\ f=r^i)\wedge(i\leq n+1))\} \Rightarrow$$

$$\{((t=\Sigma_{j=0}^{n} r^j)\wedge(\ f=r^{n+1})\wedge(i=n+1))\}$$

EJEMPLO II

- Si la postcondición es $\{(t=\Sigma_{j=0}^{n} r^j) \wedge (i>n)\}$ y el invariante:

$$\mathbf{I= \{(t=\Sigma_{j=0}^{i-1} r^j)\wedge(f=r^i)\wedge(i\leq n+1)\}}$$

$\mathbf{\{(-1\leq n)\}}$ f:=1; $\{(f=1)\wedge(-1\leq n)\}$

$\{(f=1)\wedge(-1\leq n)\}$ t:=0; $\{(t=0)\wedge (f=1)\wedge(-1\leq n)\}$

$\{(t=0)\wedge(f=1)\wedge(0<n+1)\}$ i:=0; $\mathbf{\{(t=\Sigma_{j=1}^{i-1} r^j)\wedge (f=r^i)\wedge(i\leq n+1)\}}$

while i $\leq$n do

$\mathbf{\{(t=\Sigma_{j=1}^{i-1} r^j)\wedge(f=r^i)\wedge(i<n+1)\}}$ f:=f*r; $\{(t=\Sigma_{j=1}^{i} r^j)\wedge (f=r^i)\wedge(i<n+1)\}$

$\{(t=\Sigma_{j=1}^{i} r^j)\wedge(f=r^i)\wedge(i<n+1)\}$ f:=f*r; $\{(t=\Sigma_{j=1}^{i} r^j)\wedge (f=r^{i+1})\wedge(i<n+1)\}$

$\{(t=\Sigma_{j=1}^{i} r^j)\wedge (f=r^{i+1})\wedge(i<n+1)\}$ i:=i+1 $\mathbf{\{(t=\Sigma_{j=1}^{i-1} r^j)\wedge(f=r^i)\wedge (i\leq n+1)\}}$

$$\underline{\{B \wedge I\}\ C\ \{I\}}$$

$$\{I\} \text{ while } B \text{ do } C \{(\neg B\wedge I)\}$$

$$\{(i>n)\} \Rightarrow \{i=n+1\}$$

$$\{(\neg B\wedge I)\} \Rightarrow \{(i=n+1)\wedge((t=\Sigma_{j=0}^{i-1} r^j)\wedge(f=r^i)\wedge(i\leq n+1))\} \Rightarrow$$

$$\{((t=\Sigma_{j=0}^{n} r^j)\wedge(f=r^{n+1})\wedge(i=n+1))\}$$

REGLA DE INFERENCIA PARA EL BUCLE REPEAT.

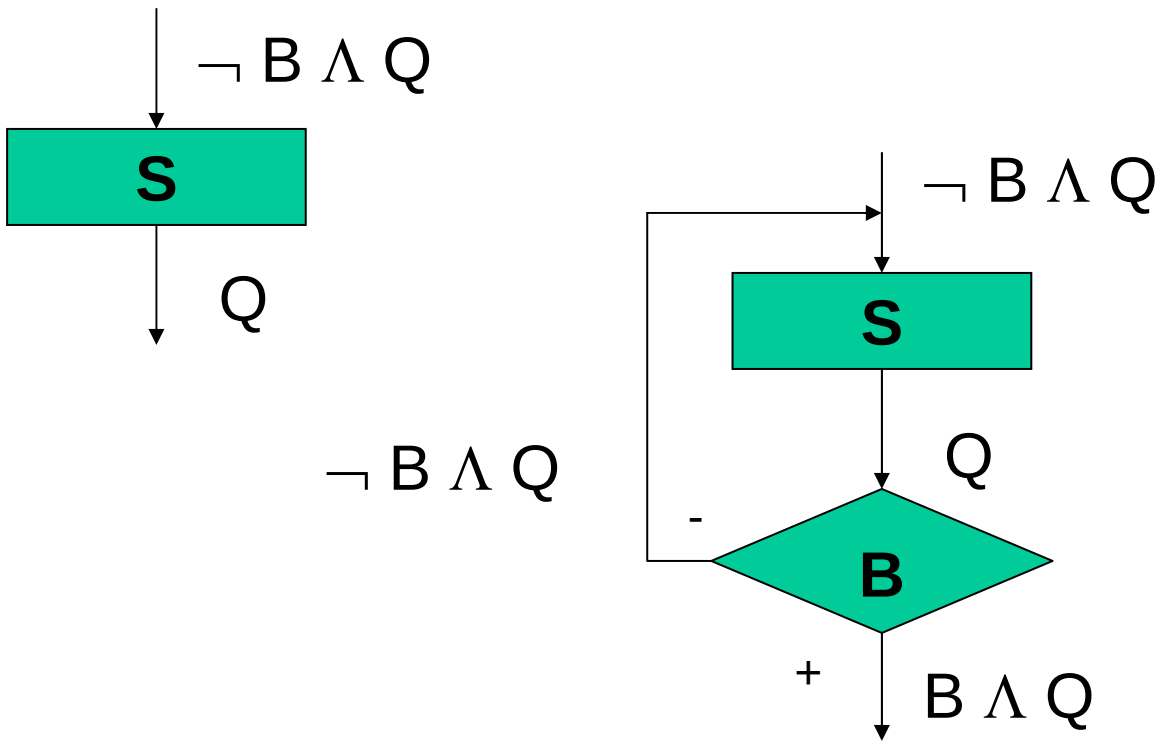
- Si C es un código tal que se cumple {I} C {Q} donde I={$\neg$ B $\wedge$ Q} entonces se puede inferir lo siguiente:

$$\frac{\{I\}\ C\ \{Q\}}{\{I\}\ \text{repeat}\ \ C\ \ \text{until B}\ \{(B \wedge Q)\}}$$

ENUNCIADO REPETITIVO TIPO REPEAT

{I} C {Q}

{I} repeat C until B {(B∧Q)}



EJEMPLO I PARA LA DETERMINACIÓN DE LA CORRECCIÓN (repeat..until)

- Determinar la corrección del siguiente código:

```
z:=0;
u:=x;
repeat
        z:=z+y;
        u:=u-1
until  u=0
```

- Se supone que este código halla el producto de dos números enteros.

EJEMPLO I

- Si la postcondición es ={(z=x*y) $\wedge$ (u=0)} y el invariante:

{Q}= {(z+u*y=x*y) $\wedge$ (u$\geq$0)}

{ P} u:=u-1;{(z+u*y=x*y) $\wedge$ (u$\geq$0)}

P={u=u-1, z+u*y=x*y, u $\geq$ 0}$\Rightarrow${z+(u-1)*y=x*y, u-1$\geq$0}

{ u-1 $\geq$ 0} $\Rightarrow$ {u $\geq$ 1} $\Rightarrow$ {u >0}

{(z+(u-1)*y=x*y) $\wedge$ (u > 0) } u:=u-1 {(z+u*y=x*y) $\wedge$ (u $\geq$ 0)}

{P} z:=z+y; {(z+(u-1)*y=x*y) $\wedge$ (u >0) }

P={z=z+y, z+(u-1)*y=x*y, u>0} $\Rightarrow$ {z+y+(u-1)*y=x*y, u >0}

{z+y+(u-1)*y=x*y} $\Rightarrow$ {z+y+u*y-y=x*y} $\Rightarrow$ {z+u*y=x*y}

{(z+u*y=x*y) $\wedge$ (u > 0)} z:=z+y; {(z+(u-1)*y=x*y) $\wedge$ (u > 0)}

Con lo que queda demostrado que {$\neg$ B $\wedge$ Q} C {Q}

{P} u:=x; {(z+u*y=x*y) $\wedge$ (u > 0)}

P={u=x, z+u*y=x*y, u>0}$\Rightarrow${z+x*y=x*y, u>0} $\Rightarrow$ {z=0, x>0}

{z=0, x>0} u:=x; {(z+u*y=x*y) $\wedge$ (u > 0)}

{P} z:=0 {(z=0) $\wedge$ (x>0)}

P={z=0, z=0, x>0}$\Rightarrow${0=0 , x>0}$\Rightarrow${x>0}

{x>0 } z:=0 {(z=0) $\wedge$ (x>0)}

{(B$\wedge$Q)} $\Rightarrow$ {(u=0) $\wedge$ ((z+u*y=x*y) $\wedge$ (u $\geq$0))} $\Rightarrow$

{(z=x*y) $\wedge$ (u=0)}

EJEMPLO I

- Si la postcondición es ={(z=x*y) $\wedge$ (u=0)} y el invariante:

I= ={$\neg$ B $\wedge$ Q}= {(z+u*y=x*y) $\wedge$ (u>0)}

Q= {(z+u*y=x*y) $\wedge$ (u $\geq$ 0)}

{x>0} z:=0 {(z=0) $\wedge$ (x>0)}

{z=0, x>0} u:=x; {(z+u*y=x*y) $\wedge$ (u > 0)}

repeat

{(z+u*y=x*y) $\wedge$ (u > 0)} z:=z+y; {(z+(u-1)*y=x*y) $\wedge$ (u > 0)}

{(z+(u-1)*y=x*y) $\wedge$ (u > 0) } u:=u-1; **{(z+u*y=x*y) $\wedge$ (u $\geq$ 0)}**

until u=0

{ I}C{Q}

{ I} repeat C until B {(B$\wedge$Q)}

{(B$\wedge$Q)} $\Rightarrow$ {(u=0) $\wedge$ ((z+u*y=x*y) $\wedge$ (u $\geq$0))} $\Rightarrow$

{(z=x*y) $\wedge$ (u=0)}

DOCUMENTACIÓN EN PSEUDOCÓDIGO

```
*****************************************
* Esta función calcula el producto de dos       *
* números  enteros   .                           *
* Argumentos: x>0                               *
* Resultado: z                                   *
*****************************************
BEGIN {Prec: x>0; Dec: u-1}
    z:=0; u:=x;
    REPEAT {z+u*y=x*y Λ u > 0}
    z:=z+y; u:= u-1
    UNTIL (u=0);
END { Post: z=x*y Λ u=0}
```

CORRECCIÓN TOTAL

- Como ya se ha definido un programa es **parcialmente correcto** si se puede demostrar que si el programa termina, lo hace satisfaciendo la postcondición final. Si a esto añadimos la demostración de que el programa finaliza, entonces se puede afirmar que es **totalmente correcto.**
- Para el subconjunto de estructuras elegidas sólo es necesario definir la finalización cuando se trabajan con bucles. En dicho caso para analizar su finalización hay que estudiar la evolución a través de las iteraciones de las variables que están involucradas en la condición de salida.
- El vector de todas las variables que aparecen en la condición de salida y que pueden ver modificado su valor durante la ejecución del bucle se denomina **vector variante**.
- Si cada secuencia de vectores variantes alcanza la condición de salida tras un número finito de iteraciones entonces el bucle termina.

EJEMPLO

- Demostrar que para n≥0 el programa siguiente termina:

```
i:=0;
f:=1;
while i<>n do
        begin
            i:=i+1;
            f:=f*r;
        end;
```

- La única variable de la condición de salida presente en el bucle es i. Por tanto el vector variante es i. El valor inicial de i es 0.
  - Si n=0 entonces no se cumple la condición de entrada y no se ejecuta el bucle. **El programa finaliza correctamente.**
  - Si n>0 entonces i alcanzará la condición de salida i=n en número finito de iteraciones igual a n ya que en cada iteración se vé incrementado en una unidad. **El programa finaliza correctamente.**

VERIFICACIÓN DE UN ALGORITMO RECURSIVO

- CORRECCIÓN PARCIAL:
  - Verificación formal de la parte del algoritmo que no contiene la llamada recursiva (inclusión de las pre y postcondiciones junto con las especificaciones de dicho algoritmo.
  - Corrección del caso Base mediante la **hipótesis de inducción.**
  - Corrección de los casos recurrentes: Verificación de las llamadas subsidiarias mediante la comprobación del **paso inductivo.**
- CORRECIÓN TOTAL:
  - Las llamadas recursivas se realizan de manera que los parámetros de las llamadas se acerquen al caso base.