Recursion and Iteration.

A recursive definition expresses the value of a function at a positive integer in terms of the values of the function at smaller integers.
This means that we can devise a recursive algorithm to evaluate a recursively defined function at a positive integer.

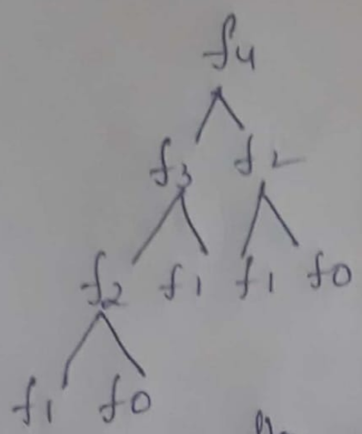
→ Instead of successively reducing the computation to the evaluation of the function at smaller integers, we can start with the value of the function at one or more integers, the base cases & successively apply the recursive definition to find the values of the function at successive larger integers.
Such a procedure is called iterative.

eg
The recursive procedure for finding fibonacci number

fib(5)
- fib(4) + fib(3)
- fib(3) + fib(2) + fib(2) + fib(1)
- fib(2) + fib(1)

Procedure fibonacci(n: nonnegative integer)
if n = 0 then fibonacci(0) = 0
Else if n = 1 then fibonacci(1) = 1
Else fibonacci(n) = fibonacci(n-1) + fibonacci(n-2).

When we use a recursive procedure to find f_n, we first express f_n as $f_{n-1} + f_{n-2}$.
Then we replace both of these fibonacci numbers by the sum of two previous fibonacci numbers & so on.
When f_1 or f_0 arises it is replaced by its value.



$f_4 = f_3 + f_2$
$f_3 = f_2 + f_1$
$f_2 = f_1 + f_0$
~~f_2~~ =
$f_2 = f_1 + f_0$

~~Note~~: ~~This~~ algorithm requires ~~...~~

Iterative approach

Procedure iterative fibonacci (n: nonnegative integer)
if $n = 0$ then $y = 0$
else
begin
$x = 0$
$y = 1$
for $i = 1$ to $n-1$
begin
$z = x + y$
$x = y$
$y = z$
End.
End.
[y is the nth fibonacci number]

0 1 1 2 3 5

fibonacci number upto 5.

$x = 0$
$y = 1$
for $i = 1$ to 4.

$i = 1$: $z = 1$, $x = 1$, $y = 1$

$i = 2$: $z = 2$, $x = 1$, $y = 2$

$i = 3$: $z = 3$, $x = 2$, $y = 3$

$i = 4$: $z = 5$, $x = 3$, $y = 5$

① Program Correctness

We need a Proof to show that the Program always gives the Correct output.

Program Verification

A Program is Said to be Correct if it Produces the Correct output for every Possible input.

A Proof that a Program is Correct Consists of two Parts. The first Part Shows that the Correct answer is obtained if the Program terminates. This Part of the Proof Establishes the Partial Correctness of the Program.

The Second Part of the Proof Shows that the Program always terminates.

To Specify what it means for a Program to Produce the Correct output, two Propositions are used. The first is the initial assertion, which gives the Properties that the input Values must have. The Second is the final assertion, which gives the Properties that the output of the Program Should have, if the Program did what was intended.

The appropriate initial & final assertions must be Provided when a Program is Checked.

Definition 1

A Program or Program Segment, S is said to be Partially Correct with respect to the initial assertion p and final assertion q if whenever p is true for the input values of S and S terminates, then q is true for the output value of S.

The notation p{S}q indicates that the Program, or Program Segment, S is Partially Correct with respect to initial assertion p and final assertion q.

Note: The notation p{S}q is known as a Hoare triple.

A simple Example illustrates the Concept of initial & final assertions.

Eg 1. Show that the Program Segment

$y := 2$
$z := x + y$ is Correct with respect to initial assertion $p: x = 1$ and final assertion $q: z = 3$.

Soln: Suppose that p is true, so that $x = 1$ as the Program begins. Then y is assigned the Value 2, and z is assigned the sum of the Values of x and y, which is 3.

Hence, S is Correct with respect to the initial assertion p and final assertion q.

Thus p{S}q is true.

(2) Rules of Inference

A useful rule of inference proves that a program is correct by splitting the program into a series of subprograms & then showing that each subprogram is correct.

Suppose that the program S is split into subprograms S_1 & S_2. Write $S = S_1; S_2$ to indicate that S is made up of S_1 followed by S_2.

Suppose that the correctness of S_1 with respect to the initial assertion p and final assertion q, & correctness of S_2 with respect to the initial assertion q and final assertion r, have been established.

It follows that if p is true & S_1 is executed & terminates, then q is true: and if q is true, & S_2 executes & terminates, then r is true.

Thus, if p is true & $S = S_1; S_2$ is executed and terminates, then r is true.

This rule of inference, called Composition rule, can be stated as

$$\frac{\begin{array}{l} p\{S_1\}q \\ q\{S_2\}r \end{array}}{\therefore p\{S_1; S_2\}r}$$

Some rules of inference for program segments involving conditional statements & loops are given. Because programs can be split into segments for proofs of correctness, this will let us verify many different programs

Conditional Statements

First, rules of inference for conditional statements will be given. Suppose that a program segment has the form

if condition then

S

where S is a block of statements. Then S is executed if condition is true & it is not executed when condition is false.

To verify that this segment is correct with respect to the initial assertion p and final assertion q, two things must be done.

First, it must be shown that when p is true & condition is also true, then q is true after S terminates.

Second, it must be shown that when p is true and condition is false, then q is true (because in this case S does not terminate).

This leads to the following rule of inference:

$(p \wedge condition)\{S\}q$

$(p \wedge \neg condition) \to q$

$\therefore p\{\text{if } condition \text{ then } S\}q$.

Example

Verify that the Program segment

③ if $x > y$ then

$y = x$.

is correct w.r.to initial assertion T and final assertion $y \geq x$

$x > y$
$3 > 2$
$y = 3$
$3 \geq 3$

Soln:- When the initial assertion is true & $x > y$, the assignment $y := x$ is carried out. Hence, the final assertion which asserts that $y \geq x$, is true in this case. Moreover, when $x > y$ is false the initial assertion is true & $x > y$ is false, so that $x \leq y$, the final assertion is again true.

$2 \geq 3$
$2 > 3$

Hence, using the rule of inference for program segments of this type, this program is correct with respect to the initial & final assertions.

$y \geq x$
$3 \geq 2$

Similarly, suppose that a Program has a Stmt of the form

if condition then
  S_1
else
  S_2.

& if condition is true, then S_1 executes; if condition is false, then S_2 executes. To verify that this Program segment is correct with respect to initial assertion p and final assertion q, 2 things must be done.

First, it must be shown that when P is true & Condition is true, then q is true after S_1 terminates. Second, it must be shown that when P is true & Condition is false, then q is true after S_2 terminates. This leads to the following rule of inference:

$$(p \wedge \text{condition}) \{S_1\} q$$
$$(p \wedge \neg \text{condition}) \{S_2\} q$$
$$\overline{\therefore\ P \{\text{if condition then } S_1 \text{ else } S_2\} q.}$$

Example Verify that the program segment

if $x<0$ then
    abs: $= -x$
else
    abs $:= x$.

is Correct with respect to the initial assertion T and final assertion abs $=|x|$.

Soln: Two things must be demonstrated. First, it must be shown that if the initial assertion is true and $x<0$, then abs $=|x|$. This is Correct, because when $x<0$ the assignment statement abs $:=-x$ sets abs $:=-x$, which is $|x|$ by definition when $x<0$. Second, it must be shown that if the initial assertion is true and $x<0$ is false, so that $x \geq 0$, then abs $=|x|$. This is also Correct, because in this program uses the assignment statement abs $:=x$, and x is $|x|$ by definition when $x \geq 0$, so abs $:=x$. Hence, using rule of inference for program segments of this type, this segment is Correct.

(4) Loop Invariants.

Proof of Correctness of While loops is described.
- To develop a rule of inference for Program segments of the type

while Condition
S

note that S is ~~(ab)~~ repeatedly Executed until Condition becomes false.

An assertion that remains true Each time S is Executed must be chosen. Such an assertion is called a loop invariant.
In otherwords, P is a loop invariant if
$(P \wedge \text{Condition})\{S\}P$ is true.
Suppose that P is a loop invariant. It follows that if P is true before the Program segment is Executed, P and $\neg$ condition are true after termination, if it occurs. This rules of inference is

$$\frac{(P \wedge \text{Condition})\{S\}P}{\therefore P\{\text{while Condition } S\}(\neg \text{condition} \wedge P).}$$

Example.

Example

A loop invariant is needed to verify that the Program Segment

```
i = 1
factorial = 1
while i < n
begin
    i = i + 1
    factorial = factorial · i
end.
```

terminates with factorial = n! when n is a Positive integer.

Let P be the assertion "factorial = i! and i ≤ n"

We first Prove that P is a loop invariant.

Suppose at the beginning of one Execution of the while loop, P is true & condition of while loop holds:

in otherwords, assume that factorial = i! and i < n.

The new Values i_{new} & $factorial_{new}$ of i and factorial are $i_{new} = i+1$ and $factorial_{new} = factorial \cdot (i+1)$

$= (i+1)! = i_{new}!$.

Because $i < n$, we also have $i_{new} = i+1 \leq n$.

Thus P is true at the End of Execution of loop. This Shows that P is a loop invariant.

Now we consider the Program Segment. Just before Entering the loop, $i = 1 \leq n$ and factorial $= 1 = 1! = i!$ both hold, So P is true. Because P is a loop invariant the rule of inference just introduced implied that if the while loop terminates, it terminate with P true & $i < n$ false.

Eg 5

i = 1
factorial = 1
while i < 5
begin
i = 2
factorial = 1·2
= 2!
i = 2 while 2 < 5
begin
i = 3
factorial = 2·3
= 6
i = 3 while 3 < 5
i = 4.
factorial 6·4

i = 4.
4 < 5
i = 5
factorial = 24·5
= 120

In this case, at the end $factorial = i!$ and $i \le n$

(5) are true, but $i < n$ is false;

in otherwords, $i = n$ and $factorial = i! = n!$ as desired.

Finally, we need to check that the while loop actually terminates. At the beginning of the Program i is assigned the value 1, So after $n-1$ traversals of the loop, the new value of i will be n, and the loop terminates at that Point.