

## CSP Program for Factorial

```
[fact (i:1 .. maxproc) ::
  *[  n: INTEGER;
    fact (i - 1) ?n →
      [  n = 0 → fact (i - 1) !1
        []  n > 0 → fact (i + 1) !n - 1;
          temp: INTEGER;
          fact (i + 1) ?temp;
          fact (i - 1) !n*temp
        ]
      ]
]
| fact (0) :: USER ]
```

## CSP Process for a Ring Buffer

```
BUF :: buffer : (0 .. 9) PORTION;
  in, out : INTEGER; in := 0; out := 0;
  *[ in < out + 10; producer ? buffer (in mod 10) → in := in + 1
    [] out < in; consumer ? more () → consumer ! buffer (out mod 10);
      out := out + 1
    ]
```

This defines the process "BUF". The producer process will include an output statement with the form

BUF ! p

to provide a value p as input to BUF, and the consumer will execute pairs of commands

BUF ! more () ; BUF ? p

An accept statement has the form

```
accept Entryname (formal parameters) do  
  Statements forming the  
  body of this entry  
end Entryname;
```

An Ada Task for Simple Message Passing.

```
task SimpleBuffer is  
  entry Place( x: in message);  
  entry Remove( x: out message);  
end;
```

  

```
task body SimpleBuffer is  
  buffer: message;  
begin  
  loop  
    accept Place( x: in message) do  
      buffer := x;  
    end Place;  
    accept Remove( x: out message) do  
      x := buffer;  
    end Remove;  
  end loop;  
end SimpleBuffer;
```

## A Bounded Buffer Using an Ada Task.

```
task BoundedBuffer is  
  entry Place(x: in message);  
  entry Remove(x: out message);  
end;  
  
task body BoundedBuffer is  
  N: constant := 10;  
  buffer: array (0..N - 1) of message;  
  i, j: integer range 0..N - 1 := 0;  
  count: integer range 0..N := 0;  
begin  
  loop  
    select  
      when count < N =>  
        accept Place (x: in message) do  
          buffer (i) := x;  
        end Place;  
        i := (i + 1) mod N; count := count + 1;  
      or  
      when count > 0 =>  
        accept Remove (x: out message) do  
          x := buffer(j);  
        end Remove;  
        i := (j + 1) mod N; count := count - 1;  
    end select;  
  end loop;  
end BoundedBuffer;
```

## Ada solution to Readers and Writers

```
task READERSANDWRITERS is
    procedure READER (READVALUE: out INTEGER);
    entry WRITER (WRITEVALUE: in INTEGER);
end;
task body READERSANDWRITERS is
    SHAREDVARIABLE: INTEGER;
    READERS: INTEGER := 0;
    entry BEGINREADING;
    entry FINISHEDREADING;
    procedure READER (READVALUE: out INTEGER) is
    begin
        BEGINREADING;
        READVALUE := SHAREDVARIABLE;
        FINISHEDREADING;

    end;
    begin
        accept WRITER (WRITEVALUE: in INTEGER) do
            SHAREDVARIABLE := WRITEVALUE;
        end;
        loop
            select
                accept BEGINREADING;
                READERS := READERS + 1;
            or
                accept FINISHEDREADING;
                READERS := READERS - 1;
            or
                when READERS = 0 =>
                    accept WRITER (WRITEVALUE: in INTEGER) do
                        SHAREDVARIABLE := WRITEVALUE;
                    end;
                end select;
            end loop;
        end READERSANDWRITERS;
```