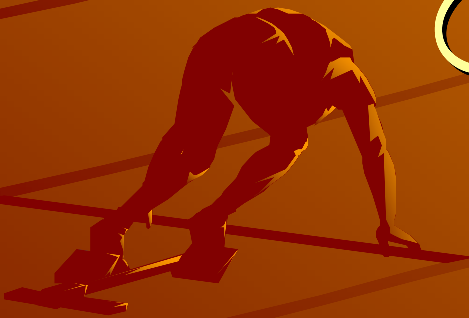
# Ideas from Communicating Sequential Processes

Prof. RKJ
CS 451 Lecture
2003



# Background of the times

- Basic Structuring methods of computer programs

  1. Repetitive construct (while, ..)
  2. Alternative command (if,...)
  3. Normal sequential program composition (;)

- Other important program structures

  subrountines
  Coroutines
  Classes
  Processes
  monitors
  Actors
  ...

# Hoare's contribution

- Components on multiprocessors must be able to communicate with each other
- What was existing: shared storage, semaphores, monitors, conditional critical regions..
- Hoare suggested that

  input and output are basic primitives of programming

  Parallel composition of sequential processes is a fundamental program structuring method

  → Development of CSP (Communicating Sequential Processes PAradigm)

# Contributions..

- Express parallel processes p1::P1 || p2::P2
- P1 and P2 can be composed together through input and output
- Inputs can be used in guards
- Guarded commands can be clubbed in an alternative statement through nondeterminism
- Repeat alternative command

# CSP Basic primitives

- Specifying parallel processes
- Communication between processes through input and output commands
- Guarded commands for obtaining nondeterminism
- Repetitive command
- Pattern matching feature for determining input messages

# Specifying parallel processes

- Specifying parallel processes
  [room::ROOM ||
  fork(i:0..4)::FORK||
  phil(0..4)::PHIL]

ROOM, FORK, PHIL provide the prcess specifications or command lists, and room, fork, and phil are the processes created

# Input and output commands

- X?i
  Input a value from X and assign it to i
- Y!10
  Output value 10 to Y
- Sem!P()
  Output signal P() to process Sem
- X(i)?V()
  Input a signal V() from the given process; refuse any other signal
- X?(x,y)
  Input a pair of values and assign them to x and y respectively

# Guarded commands and Alternative Command

Guarded commands G → CL
that is, if guard G is true, execute command list CL

- G1 → CL1 | G2 → CL2 | …. | Gn → CLn

  execute any one of the command list whose guards are found to be true: Nondeterministic choice

IF all guards fail, alternative command fails

# Repetitive command

- AS many iterations of its constituent alternative command as possible
- *[ G1 → CL1 | G2 → CL2 | .... |Gn → CLn ]

  execute any one of the command list whose guards are found to be true: Nondeterministic choice and repeat

IF all source processes fail, alternative command fails, else processes wait for corresponding outputs

# Try Bounded Buffer

# Bounded Buffer

```
[
buffer:: int item; int buff[MAX]; ....
*[

(Prod?item)and(buffernotfull)→do insertion


(consumer?consume())and(buffernotempty)→remove an item;
   consumer!removeditem
]

|| prod::PROD || consumer::CONSUMER
]
```

# Try dining philosohphers

# Dining Philosophers

[room::ROOM || fork[i:0..4]::FORK || phil[i:0..4]::PHIL]

1. ROOM

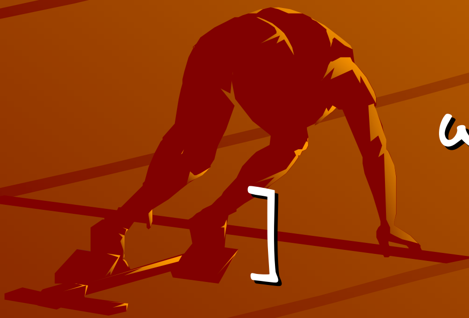
```
[int i, count;
*[(i:0..4)phil(i)?enter()andcount<4→
    count=count+1

  □(i:0..4)phil(i)?exit()→count=count-1
]
```

# Example 1

```
X :: *[
        c:char;

        west?c → east!c
  ]
```



# Example 2

```
X :: * [
        c: char;
        west?c → [
            c!='*' → east!c
          | c=='*' → west?c;
                 [c!='*'→east!'*';
                              east!c;
                 | c=='*'→east!'^'
                 ]
        ]
    ]
```

Assume that final character is not '*'

How do you deal with odd no. of '*'s at the end?

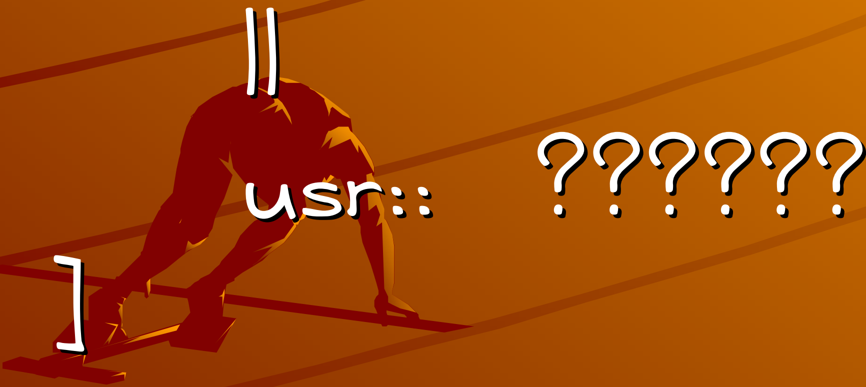
# A subroutine

[ subr:: usr?(input parameters)→

...;

X!(results)

||

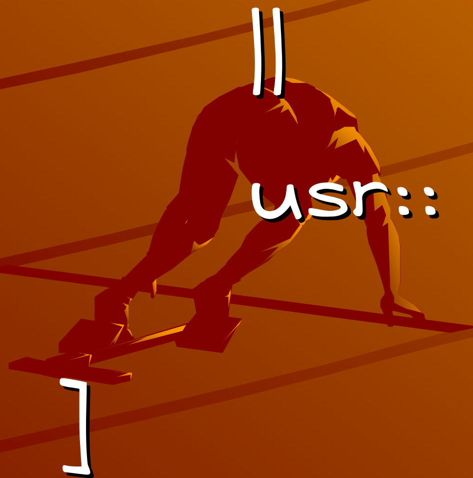


usr:: ??????

]

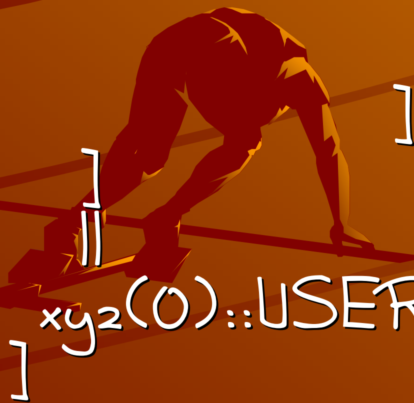
# A subroutine

```
[     subr:: usr?(input parameters)→
                      ...;
                      usr!(results)
      ||
      usr::   subr!(input params);
              subr?(results);
]
```



# Example 3

```
[xy2 (i: 1..max) ::
   *[    n:int;
         xy2(i-1)?n→
                [        n==0→xy2(i-1)!1
                       ▯ n>0→xy2(i+1)!n-1
                         r:int;
                         xy2(i+1)?r;
                         xy2(i-1)!(n*r)
                ]
   ]
   ||
   xy2(0)::USER
]
```



# Readings

- CAR Hoare, Communicating Sequential Processes, CACM, August 1978