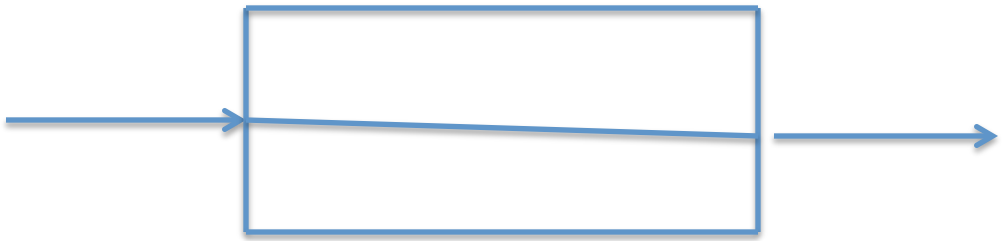
Queue - Illustration

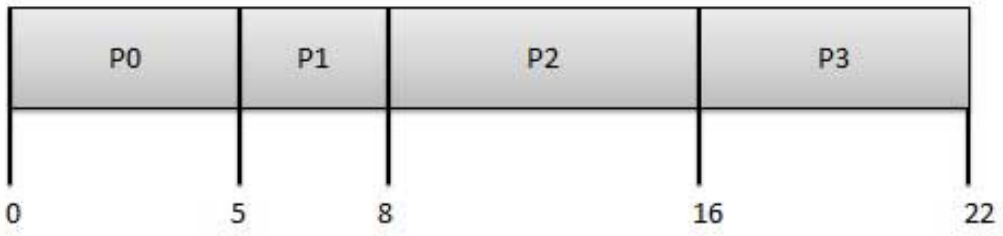
- Railway Shunting Yard Problem
- Moving the sequence of bogies to the other sections



First-Come, First-Served (FCFS) Scheduling

- Jobs are executed FCFC

Process	Arrival Time	Execute Time	Service Time
P0	0	5	0
P1	1	3	5
P2	2	8	8
P3	3	6	16



Wait time of each process

- Process Wait Time : Service Time - Arrival Time
- P0 0-0=0
- P1 5-1=4
- P2 8-2=6
- P3 16-3 = 13
- Average Wait Time: (0+4+6+13) / 4 = 5.75

Machine simulation

- Consider a machine shop that has m=3 machines and n = 4 jobs. Assume that all four jobs are available at time 0 and no new jobs are added during the simulation. Simulation continues till all jobs are completed
- M1, M2, M3 – Change over time 2, 3, 1 (waiting time after completing a task)
- Each job has several tasks (machine, time) – machine on which it is to be run and time required.
- Initially, all machines are idle
- Inactive machine – finish time is shown as L (indefinite time)

Job#	#Tasks	Tasks	Length
1	3	(1,2) (2,4) (1,1)	7
2	2	(3,4) (1,2)	6
3	2	(1,4) (2,4)	8
4	2	(3,1) (2,3)	4

(a) Job characteristics

Time	Machine Queues			Active Jobs			Finish Times		
	$M1$	$M2$	$M3$	$M1$	$M2$	$M3$	$M1$	$M2$	$M3$
Init	1,3	–	2,4	I	I	I	L	L	L
0	3	–	4	1	I	2	2	L	4
2	3	–	4	C	1	2	4	6	4
4	2	–	4	3	1	C	8	6	5
5	2	–	–	3	1	4	8	6	6
6	2,1	4	–	3	C	C	8	9	7
7	2,1	4	–	3	C	I	8	9	L
8	2,1	4,3	–	C	C	I	10	9	L
9	2,1	3	–	C	4	I	10	12	L
10	1	3	–	2	4	I	12	12	L
12	1	3	–	C	C	I	14	15	L
14	–	3	–	1	C	I	15	15	L
15	–	–	–	C	3	I	17	19	L
16	–	–	–	C	3	I	17	19	L
17	–	–	–	I	3	I	L	19	L

(b) Simulation

Job#	Finish Time	Wait Time
1	15	8
2	12	6
3	19	11
4	12	8
Total	58	33

(c) Finish and wait times

Exercise: Lists, Stacks, Queues

1. Develop a representation for integers that is suitable for performing large integers. The arithmetic is to be performed without loss of accuracy. Write methods to input and output large numbers to perform operations: add, subtract, multiply, and divide (will return quotient and remainder).

2. A **univariate polynomial** of degree d has the form

$$c_d x^d + c_{d-1} x^{d-1} + c_{d-2} x^{d-2} + \cdots + c_0$$

where $c_d \neq 0$. The c_is are the coefficients, and d, $d-1$, $\cdots$ are the exponents. By definition d is a nonnegative integer. For this exercise you may assume that the coefficients are also integers. Each $c_i x^i$ is a term of the polynomial.

For this exercise we will represent each polynomial as a linear list $(c_0, c_1, c_2, \cdots, c_d)$ of coefficients.

Develop a C++ class `polynomial` that should have an instance data member `degree`, which is the degree of the polynomial. It may have other instance data members also. Your polynomial class should support the following operations:

Develop a C++ class `polynomial` that should have an instance data member `degree`, which is the degree of the polynomial. It may have other instance data members also. Your polynomial class should support the following operations:

(a) `polynomial()`—Create the zero polynomial. The degree of this polynomial is 0 and it has no terms. `polynomial()` is the class constructor.

(b) `degree()`—Return the degree of the polynomial.

(c) `input(inStream)`—Read in a polynomial from the input stream `inStream`. You may assume the input consists of the polynomial degree and a list of coefficients in ascending order of exponents.

(d) `output(outStream)`—Output the polynomial to the output stream `outStream`. The output format should be the same as the input format.

(e) `add(b)`—Add to polynomial `b` and return the result polynomial.

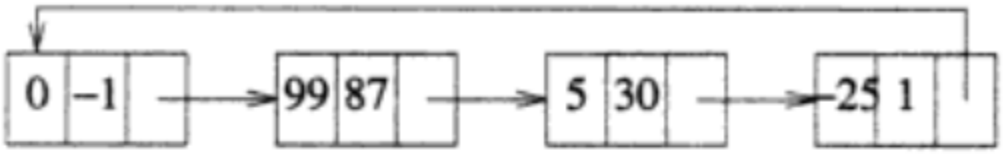
(f) `subtract(b)`—Subtract the polynomial `b` and return the result.

(g) `multiply(b)`—Multiply with polynomial `b` and return the result.

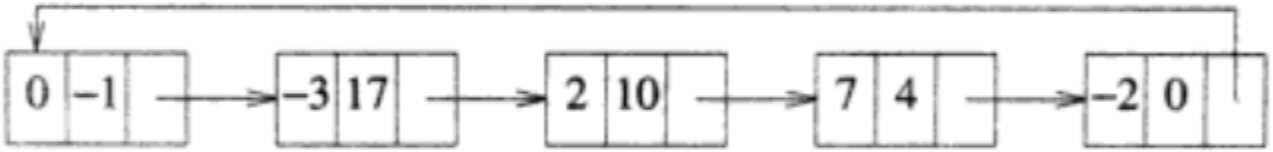
(h) `divide(b)`—Divide by polynomial `b` and return the quotient.

(i) `valueOf(x)`—Return the value of the polynomial at point `x`.

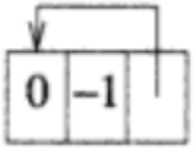
An Alternative Representation



(a) $A(x) = 99x^{87} + 5x^{30} - 25x$



(b) $B(x) = -3x^{17} + 2x^{10} + 7x^4 - 2$



(c) $C(x) = 0$

3. Using this data structure solve Question 2

4. Write a simulation program for the job description as illustrated.

Big Endian Vs Little Endian

- Modern computers store one byte of data in each memory address or location, i.e., byte addressable memory.
  - An 32-bit integer is, therefore, stored in 4 memory addresses.
- The term"Endian" refers to the *order* of storing bytes in computer memory. In "Big Endian" scheme, the most significant byte is stored first in the lowest memory address (or big in first),
- while "Little Endian" stores the least significant bytes in the lowest memory address.
- For example, the 32-bit integer 12345678H (2215053170_{10}) is stored as 12H 34H 56H 78H in big endian
- and 78H 56H 34H 12H in little endian.
- An 16-bit integer 00H 01H is interpreted as 0001H in big endian, and 0100H as little endian.