CS 213 Tutorial

Mayukh Rath
Amit Goyal

Scheduling : FCFS

Example:-

Process	Burst Time
P_1	24
P_2	3
P_3	3

Average Waiting Time = (0+24+27)/3 =17

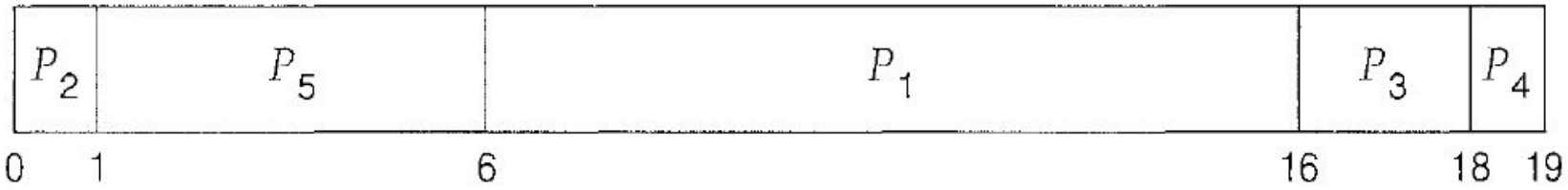
Program for FCFS

```
for(i=1;i<n;i++)
{
     wt[i]=0;
     for(j=0;j<i;j++)
          wt[i]+=bt[j];
}
```

```
for(i=0;i<n;i++)
{
    tat[i]=bt[i]+wt[i];
}
```

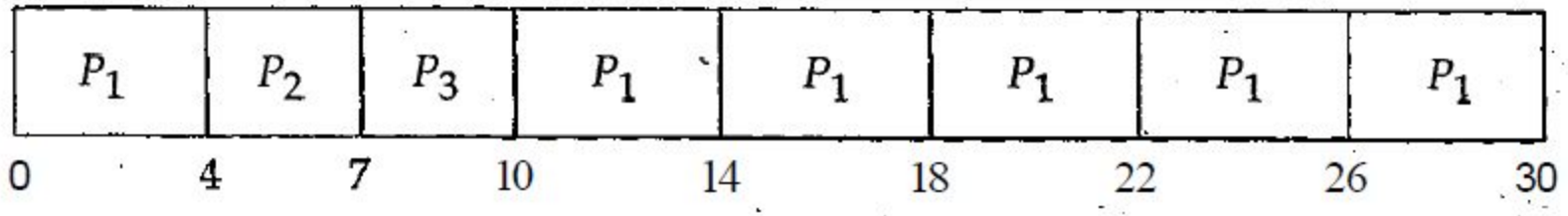
Priority Based Scheduling

Process	Burst Time	Priority
P_1	10	3
P_2	1	1
P_3	2	4
P_4	1	5
P_5	5	2



Average Waiting Time - 8.2 milliseconds

Round Robin Scheduling



The average waiting time is 17/3 = 5.66 milliseconds.

Machine Simulation Problem

- 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)

Machine Simulation Problem

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

Machine Simulation Problem

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

Machine Simulation Problem

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

Machine Simulation Problem

```
void main()
{
   inputData();          // get machine and job data
   startShop();          // initial machine loading
   simulate();           // run all jobs through shop
   outputStatistics();   // output machine wait times
}
```

Machine Simulation Problem

```
struct task
{
   int machine;
   int time;

   task(int theMachine = 0, int theTime = 0)
   {
      machine = theMachine;
      time = theTime;
   }
};
```

```
struct job
{
   arrayQueue<task> taskQ;    // this job's tasks
   int length;                // sum of scheduled task times
   int arrivalTime;           // arrival time at current queue
   int id;                    // job identifier

   job(int theId = 0)
   {
      id = theId;
      length = 0;
      arrivalTime = 0;
   }

   void addTask(int theMachine, int theTime)
   {
      task theTask(theMachine, theTime);
      taskQ.push(theTask);
   }

   int removeNextTask()
   {// Remove next task of job and return its time.
    // Also update length.

      int theTime =  taskQ.front().time;
      taskQ.pop();
      length += theTime;
      return theTime;
   }
};
```

Machine Simulation Problem

```
struct machine
{
   arrayQueue<job*> jobQ;
                    // queue of waiting jobs for this machine
   int changeTime;  // machine change-over time
   int totalWait;   // total delay at this machine
   int numTasks;    // number of tasks processed on this machine
   job* activeJob;  // job currently active on this machine

   machine()
   {
      totalWait = 0;
      numTasks = 0;
      activeJob = NULL;
   }
};
```

```
class eventList
{
   public:
      eventList(int theNumMachines, int theLargeTime)
      {// Initialize finish times for n machines.
         if (theNumMachines < 1)
            throw illegalParameterValue
                  ("number of machines must be >= 1");
         numMachines = theNumMachines;
         finishTime = new int [numMachines + 1];

         // all machines are idle, initialize with
         // large finish time
         for (int i = 1; i <= numMachines; i++)
            finishTime[i] = theLargeTime;
      }

      int nextEventMachine()
      {// Return machine for next event.

         // find first machine to finish, this is the
         // machine with smallest finish time
         int p = 1;
         int t = finishTime[1];
         for (int i = 2; i <= numMachines; i++)
            if (finishTime[i] < t)
            {// i finishes earlier
               p = i;
               t = finishTime[i];
            }
         return p;
      }

      int nextEventTime(int theMachine)
      {return finishTime[theMachine];}

      void setFinishTime(int theMachine, int theTime)
      {finishTime[theMachine] = theTime;}
   private:
      int* finishTime;    // finish time array
      int numMachines;    // number of machines
};
```

Machine Simulation Problem

```
// global variables
int timeNow;              // current time
int numMachines;          // number of machines
int numJobs;              // number of jobs
eventList* eList;         // pointer to event list
machine* mArray;          // array of machines
int largeTime = 10000;    // all machines finish before this
```

```
void inputData()
{// Input machine shop data.

   cout << "Enter number of machines and jobs" << endl;
   cin >> numMachines >> numJobs;
   if (numMachines < 1 || numJobs < 1)
         throw illegalInputData
               ("number of machines and jobs must be >= 1");

   // create event and machine queues
   eList = new eventList(numMachines, largeTime);
   mArray = new machine [numMachines + 1];

   // input the change-over times
   cout << "Enter change-over times for machines" << endl;
   int ct;
   for (int j = 1; j <= numMachines; j++)
   {
      cin >> ct;
      if (ct < 0)
         throw illegalInputData("change-over time must be >= 0");
      mArray[j].changeTime = ct;
   }
```

```
    // input the jobs
    job* theJob;
    int numTasks, firstMachine, theMachine, theTaskTime;
    for (int i = 1; i <= numJobs; i++)
    {
       cout << "Enter number of tasks for job " << i << endl;
       cin >> numTasks;
       firstMachine = 0;   // machine for first task
       if (numTasks < 1)
          throw illegalInputData("each job must have > 1 task");

       // create the job
       theJob = new job(i);
       cout << "Enter the tasks (machine, time)"
            << " in process order" << endl;
       for (int j = 1; j <= numTasks; j++)
       {// get tasks for job i
          cin >> theMachine >> theTaskTime;
          if (theMachine < 1 || theMachine > numMachines
              || theTaskTime < 1)
          throw illegalInputData("bad machine number or task time");
          if (j == 1)
             firstMachine = theMachine;    // job's first machine
          theJob->addTask(theMachine, theTaskTime);  // add to
       }                                             // task queue
       mArray[firstMachine].jobQ.push(theJob);
    }
}
```

```
void startShop()
{// Load first jobs onto each machine.
   for (int p = 1; p <= numMachines; p++)
      changeState(p);
}
```

```
job* changeState(int theMachine)
{// Task on theMachine has finished, schedule next one.
 // Return last job run on this machine.
   job* lastJob;
   if (mArray[theMachine].activeJob == NULL)
   {// in idle or change-over state
      lastJob = NULL;
      // wait over, ready for new job
      if (mArray[theMachine].jobQ.empty()) // no waiting job
            eList->setFinishTime(theMachine, largeTime);
      else
      {// take job off the queue and work on it
         mArray[theMachine].activeJob =
               mArray[theMachine].jobQ.front();
         mArray[theMachine].jobQ.pop();
         mArray[theMachine].totalWait +=
              timeNow - mArray[theMachine].activeJob->arrivalTime;
         mArray[theMachine].numTasks++;
         int t = mArray[theMachine].activeJob->removeNextTask();
         eList->setFinishTime(theMachine, timeNow + t);
      }
   }
   else
   {// task has just finished on theMachine
    // schedule change-over time
      lastJob = mArray[theMachine].activeJob;
      mArray[theMachine].activeJob = NULL;
      eList->setFinishTime(theMachine, timeNow +
                           mArray[theMachine].changeTime);
   }

   return lastJob;
}
```

```
void simulate()
{// Process all jobs to completion.
   while (numJobs > 0)
   {// at least one job left
      int nextToFinish = eList->nextEventMachine();
      timeNow = eList->nextEventTime(nextToFinish);
      // change job on machine nextToFinish
      job* theJob = changeState(nextToFinish);
      // move theJob to its next machine
      // decrement numJobs if theJob has finished
      if (theJob != NULL && !moveToNextMachine(theJob))
         numJobs--;
   }
}
```

```
bool moveToNextMachine(job* theJob)
{// Move theJob to machine for its next task.
 // Return false iff no next task.

   if (theJob->taskQ.empty())
   {// no next task
      cout << "Job " << theJob->id << " has completed at "
           << timeNow << " Total wait was "
           << (timeNow - theJob->length) << endl;
      return false;
   }
   else
   {// theJob has a next task
    // get machine for next task
      int p = theJob->taskQ.front().machine;
      // put on machine p's wait queue
      mArray[p].jobQ.push(theJob);
      theJob->arrivalTime = timeNow;
      // if p idle, schedule immediately
      if (eList->nextEventTime(p) == largeTime)
         // machine is idle
         changeState(p);

      return true;
   }
}
```

Machine Simulation Problem

```
void outputStatistics()
{// Output wait times at machines.
   cout << "Finish time = " << timeNow << endl;
   for (int p = 1; p <= numMachines; p++)
   {
      cout << "Machine " << p << " completed "
           << mArray[p].numTasks << " tasks" << endl;
      cout << "The total wait time was "
           << mArray[p].totalWait << endl;
      cout <<  endl;
   }
}
```

Machine Simulation Problem

Write an enhanced machine shop simulator that allows jobs to enter the shop during simulation. The simulation stops at a specified time. Jobs that have not been completed by this time remain incomplete.

Large Integer Problem

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).

Multiply Function

```
string multiply(string num1, string num2) {
        int len1 = num1.length();
        int len2 = num2.length();
        if(len1 < len2){
            string temp = num1;
            num1 = num2;
            num2 = temp;
            len2 = len1;
        }
        int i = len2 - 1 ;
        string ans = "0";
        while(i>=0){
            string t =
mult(num1,num2.at(i));
            cout << "t " <<t <<endl;
            ans = sum1(ans,t,len2 -1 - i);
            cout << "ans " <<  ans << endl;
            --i;
        }
```

```
int s = 0;
       bool flag = true;
       while(s < ans.length()){
           if(ans.at(s) != '0'){
               flag = false;
               break;
           }
           ++s;
       }
       if(flag){
           return "0";
       }
       return ans;
   }
```

Mult Function

```
string mult(string str , char c){
        string ans;
        int len = str.length();
        int i = len - 1;
        int q = 0, remaind = 0;
        while(i >= 0){
            q = (str.at(i) - '0') * (c - '0') + remaind;
            remaind = q / 10;
            ans = to_string(q%10).append(ans);
            --i;
        }
        if(remaind > 0){
           ans = to_string(remaind).append(ans);
        }
        return ans;
    }
```

Sum Function

```
string sum1(string str1 , string str2, int pos){
        int i = str1.length() - 1 - pos ;
        int j = str2.length()-1;
        string ans = str1.substr(i+1);
        int q = 0, remaind = 0;
        while(i >=0 && j >=0){
            q = (str1.at(i) - '0') + (str2.at(j)
- '0') + remaind;
            remaind = q /10;
            ans = to_string(q%10).append(ans);
            --i;
            --j;
       }
        string temp;
        if(i == -1 && j == -1){
        }
else{
            if(i == -1){
               temp = str2.substr(0,j+1);
            }else{
               temp = str1.substr(0,i+1);
            }
            i = temp.length() -1;
            while(i >=0){
                q = temp.at(i) - '0' + remaind;
                remaind = q /10;
                ans =
to_string(q%10).append(ans);
                --i;
            }
        }
        if(remaind > 0){
                ans =
to_string(remaind).append(ans);
        }
        return ans;
}
```

Polynomial Class Implementation

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`.

Class Structures

```
public class LinkedPolynomial
{
int degree;
Node first;
Node last;
private class Node
{
public int coeff;
public Node next;
public Node()
{
coeff=0;
next=null;
}
```

Member Functions:

```
public void input(Scanner in)
{
degree=in.nextInt();
for(int i=0;i<=degree;i++)
{
if(i==0)
{
first=new Node(in.nextInt());
last=first;
}
else
{last.next=new Node(in.nextInt());
last=last.next;
}
}
}
```

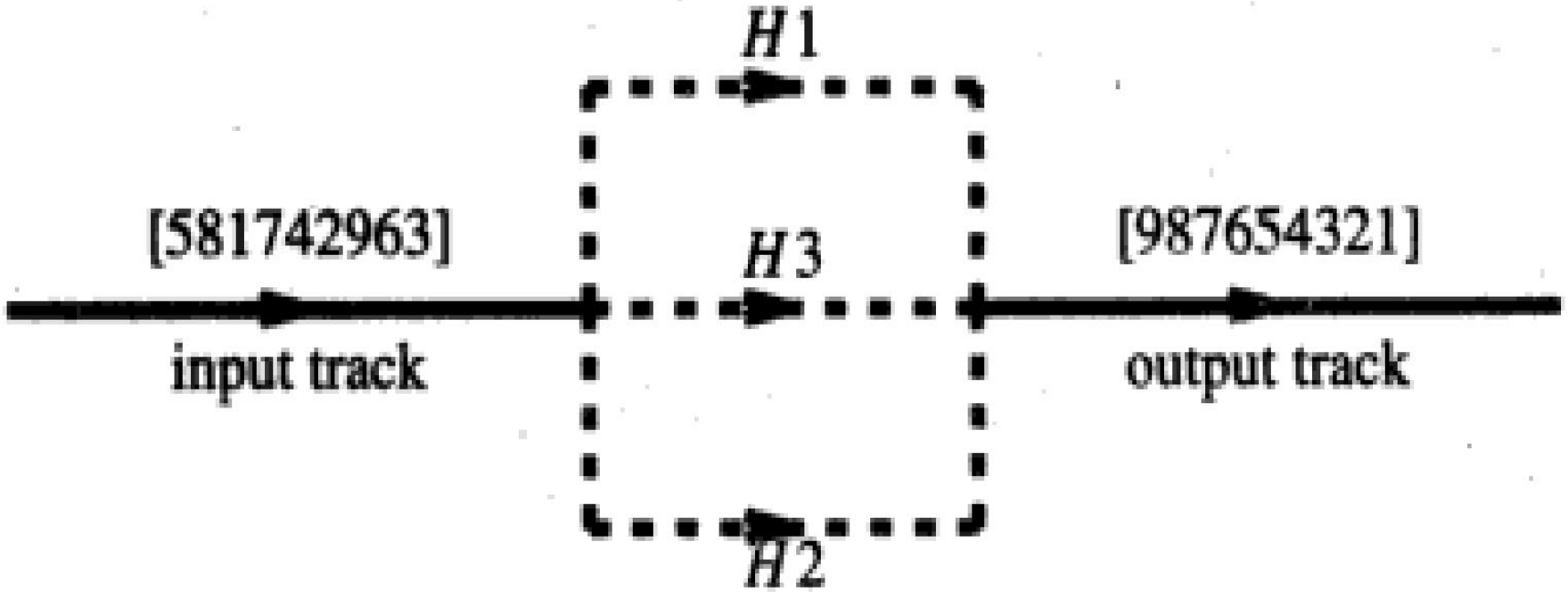
```
public LinkedPolynomial add(LinkedPolynomial b)
{
LinkedPolynomial a=this;
LinkedPolynomial p= new LinkedPolynomial(Math.max(this.getDegree(),b.getDegree()));
for(int i=0;i<=p.getDegree();i++)
{
if(i==0)
{
p.first=new Node();
p.last=p.first;
}
else
{
Node cur=new Node();
p.last.next=cur;
p.last=cur;
}
}

Node current1=p.first;
Node current2=this.first;
for(int i=0;i<=this.getDegree();i++)
{
current1.coeff+=current2.coeff;
current1=current1.next;
current2=current2.next;
}
current1=p.first;
current2=b.first;
for(int i=0;i<=b.getDegree();i++)
{
current1.coeff+=current2.coeff;
current1=current1.next;
current2=current2.next;
}

return p;
}
```

```
public LinkedPolynomial multiply(LinkedPolynomial b)
{
LinkedPolynomial a = this;
LinkedPolynomial c = new LinkedPolynomial();
 for (int i=0;i<this.getDegree();i++)
{
    Node x=this.first;
   LinkedPolynomial temp = new LinkedPolynomial();
for (int j=0;j<this.getDegree();j++)
{
Node y=b.first;
   if(i==0 && j==0)
{
temp.first=new Node(x.coeff * y.coeff);
temp.last=temp.first;
}
else
{
 temp.last.next = new Node(x.coeff * y.coeff);
temp.last = temp.last.next;
     }
}
 c = c.add(temp);
    }
        return c;
}
}
```

Railway Shunting Yard Problem



```
private static void outputFromHoldingTrack()
{
   // remove smallestCar from itsTrack
   track[itsTrack].remove();
   System.out.println("Move car " + smallestCar + " from holding "
                      "track " + itsTrack + " to output track");

   // find new smallestCar and itsTrack by checking all queue fronts
   smallestCar = numberOfCars + 2;
   for (int i = 1; i <= numberOfTracks; i++)
      if (!track[i].isEmpty() &&
      ((Integer) track[i].getFrontElement()).intValue() < smallestCar)
      {
         smallestCar = ((Integer) track[i].getFrontElement())
                                          .intValue();
         itsTrack = i;
      }
}
```

```
private static boolean putInHoldingTrack(int c)
{
   // find best holding track for car c
   // initialize
   int bestTrack = 0,  // best track so far
       bestLast =  0;  // last car in bestTrack

   // scan tracks
   for (int i = 1; i <= numberOfTracks; i++)
      if (!track[i].isEmpty())
      {// track i not empty
          int lastCar = ((Integer) track[i].getRearElement())
                                                 .intValue();
          if (c > lastCar && lastCar > bestLast)
          {
             // track i has bigger car at its rear
             bestLast = lastCar;
             bestTrack = i;
          }
      }
      else // track i empty
         if (bestTrack == 0) bestTrack = i;

   if (bestTrack == 0) return false; // no feasible track

   // add c to bestTrack
   track[bestTrack].put(new Integer(c));
   System.out.println("Move car " + c + " from input track "
                      + "to holding track " + bestTrack);

   // update smallestCar and itsTrack if needed
   if (c < smallestCar)
   {   smallestCar = c;
       itsTrack = bestTrack;
   }

   return true;
}
```

Problem:-

(a) You have a railroad shunting yard with three shunting tracks that operate as queues. The initial ordering of cars is 3, 1, 7, 6, 2, 8, 5, 4. Draw figures similar to Figures 10.11 to show the configuration of the shunting tracks, the input track, and the output track following each car move made by the solution of Section 10.5.1.

(b) Do part (a) for the case when the number of shunting tracks is 2.

Example Of Iterator

```
public interface Iterator
{
    boolean hasNext( );
    Object next( );
}
```

```
public class MyContainer
{
    Object [ ] items;
    int size;

    public Iterator iterator( )
      { return new MyContainerIterator( this ); }

    // Other methods not shown.
}
```

```
class MyContainerIterator implements Iterator
{
    private int current = 0;
    private MyContainer container;

    MyContainerIterator( MyContainer c )
      { container = c; }

    public boolean hasNext( )
      { return current < container.size; }

    public Object next( )
      { return container.items[ current++ ]; }
}
```

Example Of Iterator

```
public static void main( String [ ] args )
{
    MyContainer v = new MyContainer( );

    v.add( "3" );
    v.add( "2" );

    System.out.println( "Container contents: " );
    Iterator itr = v.iterator( );
    while( itr.hasNext( ) )
        System.out.println( itr.next( ) );
}
```

Big Endian and Little Endian

Storing Words in Memory:

Ex:- 90AB12CD

Big Endian

Address	Value
1000	90
1001	AB
1002	12
1003	CD

Little Endian

Address	Value
1000	CD
1001	12
1002	AB
1003	90

Programs:- Actual Memory Representation in Memory

```
#include <stdio.h>

/* function to show bytes in memory, from location start to start+n*/
void show_mem_rep(char *start, int n)
{
    int i;
    for (i = 0; i < n; i++)
            printf(" %.2x", start[i]);
    printf("\n");
}

/*Main function to call above function for 0x01234567*/
int main()
{
int i = 0x01234567;
show_mem_rep((char *)&i, sizeof(i));
getchar();
return 0;
}
```

Find Endianness of Machine

```
#include <stdio.h>
int main()
{
   unsigned int i = 1;
   char *c = (char*)&i;
   if (*c)
      printf("Little endian");
   else
      printf("Big endian");
   getchar();
   return 0;
}
```