Name: Varun Menon **Course Code: CSE4001**

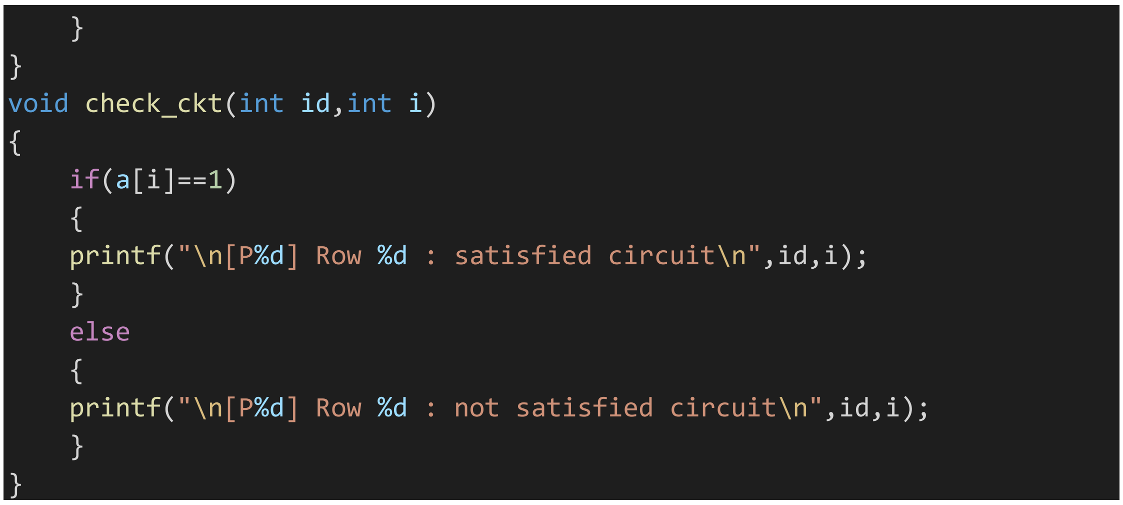
Reg No: 19BCE1438 **Faculty: Dr. Harini S**

Lab Experiment 9

1. Circuit Satisfiability

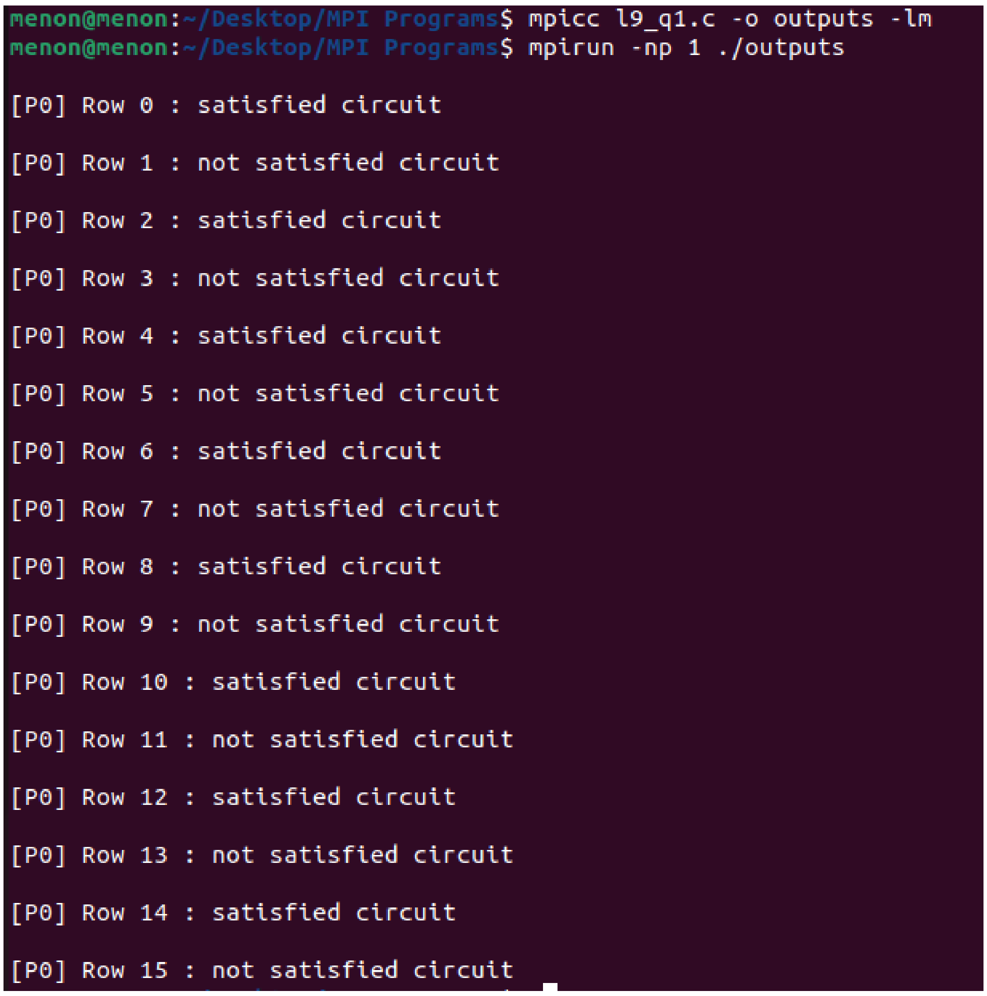
   Code:

```
#include<mpi.h>
#include<stdio.h>
#include<math.h>

void generate_tt(int);
void check_ckt(int,int);
int a[16];

int main()
{
    MPI_Init(NULL,NULL);
    int i, n=4;
    int id;
    int p;
    MPI_Comm_rank(MPI_COMM_WORLD, &id);
    MPI_Comm_size(MPI_COMM_WORLD, &p);
    int m = pow(2,n);
    generate_tt(m);
    for(int i=id;i<m;i+=p)
    {
        check_ckt(id,i);
    }
    MPI_Finalize();
    return 0;
}

void generate_tt(int m)
{
    for(int i=0;i<m;i++)
    {
        if(i%2==0)
        {
            a[i]=1;
        }
        else
        {
            a[i]=0;
        }
```



```
    }
}
void check_ckt(int id,int i)
{
    if(a[i]==1)
    {
    printf("\n[P%d] Row %d : satisfied circuit\n",id,i);
    }
    else
    {
    printf("\n[P%d] Row %d : not satisfied circuit\n",id,i);
    }
}
```

Output:



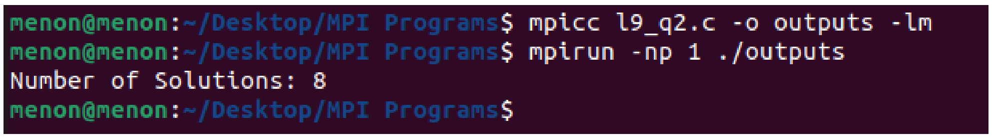
```
menon@menon:~/Desktop/MPI Programs$ mpicc l9_q1.c -o outputs -lm
menon@menon:~/Desktop/MPI Programs$ mpirun -np 1 ./outputs

[P0] Row 0 : satisfied circuit

[P0] Row 1 : not satisfied circuit

[P0] Row 2 : satisfied circuit

[P0] Row 3 : not satisfied circuit

[P0] Row 4 : satisfied circuit

[P0] Row 5 : not satisfied circuit

[P0] Row 6 : satisfied circuit

[P0] Row 7 : not satisfied circuit

[P0] Row 8 : satisfied circuit

[P0] Row 9 : not satisfied circuit

[P0] Row 10 : satisfied circuit

[P0] Row 11 : not satisfied circuit

[P0] Row 12 : satisfied circuit

[P0] Row 13 : not satisfied circuit

[P0] Row 14 : satisfied circuit

[P0] Row 15 : not satisfied circuit
```

2. Number of solutions in circuit satisfiability

Code:

```c
#include<mpi.h>
#include<stdio.h>
#include<math.h>

void generate_tt(int);
void check_ckt(int,int);
int a[16], count;

int main()
{
    MPI_Init(NULL,NULL);
    int i, n=4;
    int id;
    int p;
    MPI_Comm_rank(MPI_COMM_WORLD, &id);
    MPI_Comm_size(MPI_COMM_WORLD, &p);
    int m = pow(2,n);
    generate_tt(m);
    for(int i=id;i<m;i+=p)
    {
        check_ckt(id,i);
    }
    printf("Number of Solutions: %d\n", count);
    MPI_Finalize();
    return 0;
}

void generate_tt(int m)
{
    for(int i=0;i<m;i++)
    {
        if(i%2==0)
        {
            a[i]=1;
        }
        else
        {
            a[i]=0;
        }
    }
}
void check_ckt(int id,int i)
{
    if(a[i]==1)
    {
        count++;
    }
}
```

Output:



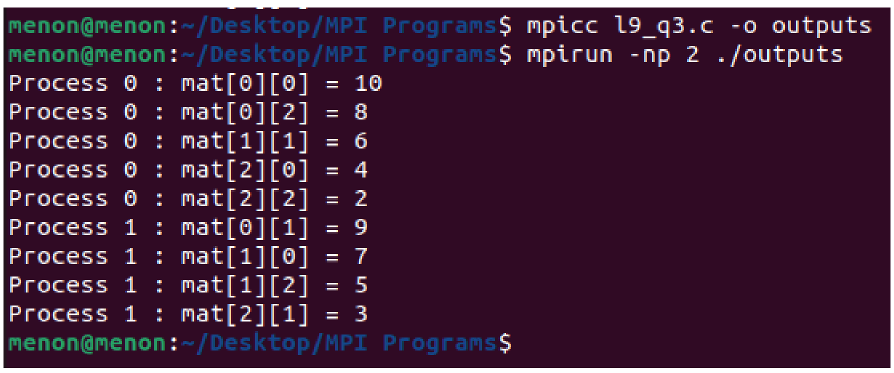
```
menon@menon:~/Desktop/MPI Programs$ mpicc l9_q2.c -o outputs -lm
menon@menon:~/Desktop/MPI Programs$ mpirun -np 1 ./outputs
Number of Solutions: 8
menon@menon:~/Desktop/MPI Programs$
```

3. Adding a count to all values of a matrix with size n*n

   Code:

```
#include <mpi.h>
#include <stdio.h>

int main(int argc, char** argv) {
    int p, id;
    int n = 3, counter = 1;
    int mat[3][3] = {{9,8,7},
        {6,5,4},
        {3,2,1}};
    MPI_Init(NULL, NULL);
    MPI_Comm_size(MPI_COMM_WORLD, &p);
    MPI_Comm_rank(MPI_COMM_WORLD, &id);
    for(int x = id; x < n*n; x += p)
    {
        int i = x / n, j = x % n;
        mat[i][j] += counter;
        printf("Process %d : mat[%d][%d] = %d\n", id, i, j, mat[i][j]);
    }
    MPI_Finalize();
    return 0;
}
```

Output:



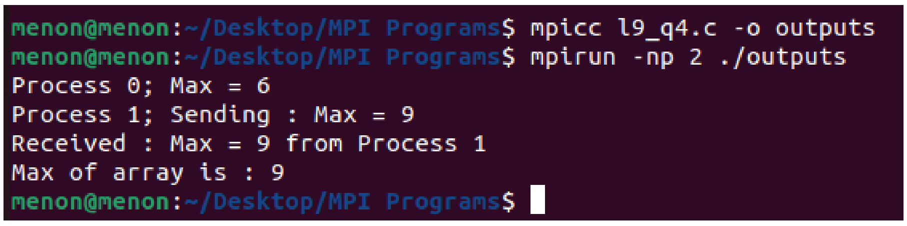
```
menon@menon:~/Desktop/MPI Programs$ mpicc l9_q3.c -o outputs
menon@menon:~/Desktop/MPI Programs$ mpirun -np 2 ./outputs
Process 0 : mat[0][0] = 10
Process 0 : mat[0][2] = 8
Process 0 : mat[1][1] = 6
Process 0 : mat[2][0] = 4
Process 0 : mat[2][2] = 2
Process 1 : mat[0][1] = 9
Process 1 : mat[1][0] = 7
Process 1 : mat[1][2] = 5
Process 1 : mat[2][1] = 3
menon@menon:~/Desktop/MPI Programs$
```

4. Find Max of 'n' no's

Code:

```
#include <mpi.h>
#include <stdio.h>
#include<math.h>
#include<limits.h>

int main(int argc, char** argv) {
    int p, id, n = 10;
    int arr[10] = {2, 5, 6, 9, 4, 7, 1, 8, 2, 1};

    MPI_Request request;
    MPI_Status status;
    MPI_Init(NULL, NULL);
    MPI_Comm_size(MPI_COMM_WORLD, &p);
    MPI_Comm_rank(MPI_COMM_WORLD, &id);
    int m = INT_MIN;

    for(int i = id; i < n; i += p)
    {
        if(arr[i] > m)
        {
            m = arr[i];
        }
    }

    if(id != 0)
    {
        printf("Process %d; Sending : Max = %d\n", id, m);
        MPI_Isend(&m, 1, MPI_INT, 0, 123, MPI_COMM_WORLD, &request);
        MPI_Wait(&request, &status);
    }
    else
    {
        printf("Process 0; Max = %d\n", m);
        for(int j = 1; j < p; j++)
        {
            int x;
            MPI_Request requestj;
            MPI_Irecv(&x, 1, MPI_INT, j, 123, MPI_COMM_WORLD,
&requestj);
            MPI_Wait(&requestj, &status);
        printf("Received : Max = %d from Process %d\n", x, j);
            if(x > m) m = x;
        }
        printf("Max of array is : %d\n", m);
    }
    MPI_Finalize();
    return 0;
}
```

Output:

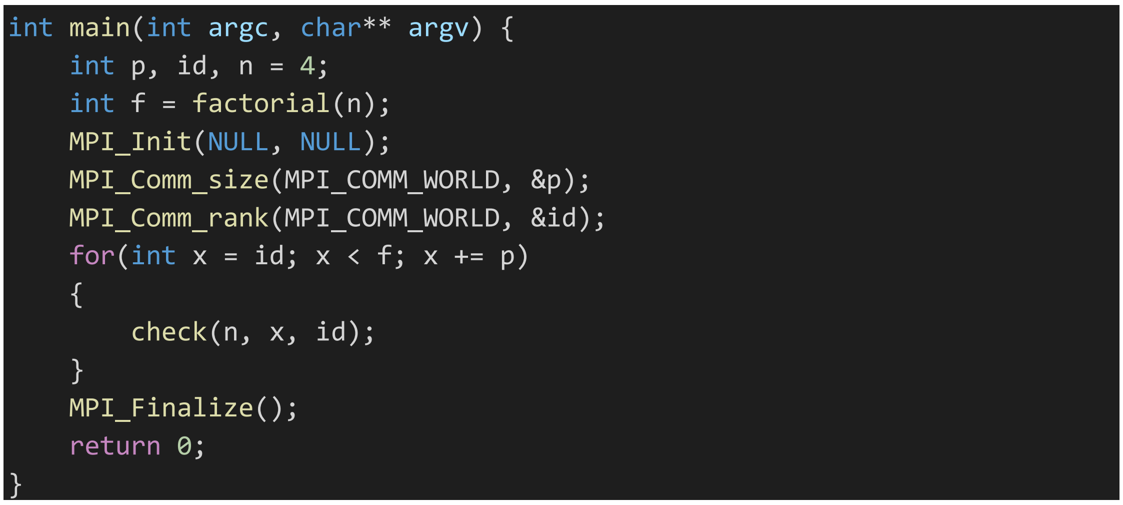


```
menon@menon:~/Desktop/MPI Programs$ mpicc l9_q4.c -o outputs
menon@menon:~/Desktop/MPI Programs$ mpirun -np 2 ./outputs
Process 0; Max = 6
Process 1; Sending : Max = 9
Received : Max = 9 from Process 1
Max of array is : 9
menon@menon:~/Desktop/MPI Programs$ 
```

5. Four Queen's Problem

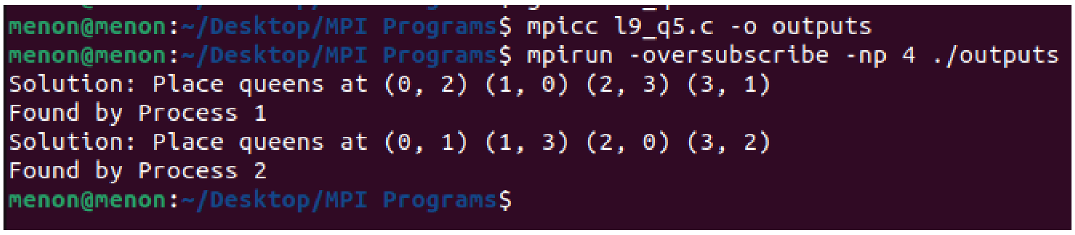
Code:

```c
#include <mpi.h>
#include <stdio.h>
int factorial(int n)
{
    int f = 1;
    for(int i = 2; i < n+1; i++)
    {
        f *= i;
    }
    return f;
}
int check(int n, int k, int id)
{
    int len = n;
    int perm[n];
    int nums[n];
    for(int i = 0; i < n; i++) nums[i] = i;
    int size = n;
    for(int i = 0; i < n; i++)
    {
        int index = k / factorial(n-i-1);
        perm[i] = nums[index];
        for(int j = index; j < size-1; j++)
        {
            nums[j] = nums[j+1];
            size--;
        }
        k -= index*factorial(n-i-1);
    }

    for(int i = 0; i < n; i++)
    {
        for(int j = i+1; j < n; j++)
        {
            if ((i-perm[i] == j-perm[j]) || (i+perm[i] == j+perm[j]))
            {
                return 0;
            }
        }
    }
    printf("Process %d: found solution place queens at: (%d, %d) (%d,
%d) (%d, %d) (%d, %d)\n", id, 0, perm[0], 1, perm[1], 2, perm[2], 3,
perm[3]);
    return 1;
}
```



```c
int main(int argc, char** argv) {
    int p, id, n = 4;
    int f = factorial(n);
    MPI_Init(NULL, NULL);
    MPI_Comm_size(MPI_COMM_WORLD, &p);
    MPI_Comm_rank(MPI_COMM_WORLD, &id);
    for(int x = id; x < f; x += p)
    {
        check(n, x, id);
    }
    MPI_Finalize();
    return 0;
}
```

Output:



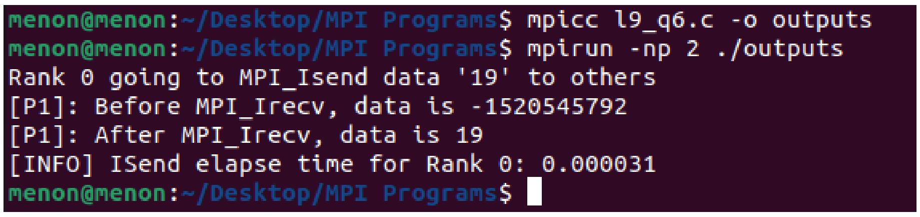
```
menon@menon:~/Desktop/MPI Programs$ mpicc l9_q5.c -o outputs
menon@menon:~/Desktop/MPI Programs$ mpirun -oversubscribe -np 4 ./outputs
Solution: Place queens at (0, 2) (1, 0) (2, 3) (3, 1)
Found by Process 1
Solution: Place queens at (0, 1) (1, 3) (2, 0) (3, 2)
Found by Process 2
menon@menon:~/Desktop/MPI Programs$
```

6. Sample isend, ireceive with mpi_wtime

Code:

```c
#include <mpi.h>
#include <stdio.h>
int main(int argc, char** argv)
{
    int rank;
    int buf;
    int world_size;
    const int root=0;

    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &world_size);
    MPI_Status status;
    MPI_Request request = MPI_REQUEST_NULL;
    double time = 0;

    if(rank == root)
    {
        buf = 19;
        printf("Rank %d going to MPI_Isend data '%d' to
others\n",rank,buf);
        time = - MPI_Wtime();
        for(int i=0;i<world_size;i++)
        {
            MPI_Isend(&buf, 1, MPI_INT, i, 0, MPI_COMM_WORLD,&request);
        }
        time += MPI_Wtime();
    }
    else
    {
        printf("[P%d]: Before MPI_Irecv, data is %d\n", rank, buf);
        MPI_Irecv(&buf, 1, MPI_INT, 0, 0, MPI_COMM_WORLD,&request);
        MPI_Wait(&request, &status);
        printf("[P%d]: After MPI_Irecv, data is %d\n", rank, buf);
    }
    MPI_Finalize();
    if(rank == root)
    {
        printf("[INFO] ISend elapse time for Rank %d: %f\n",rank,time);
    }
    return 0;
}
```

Output:



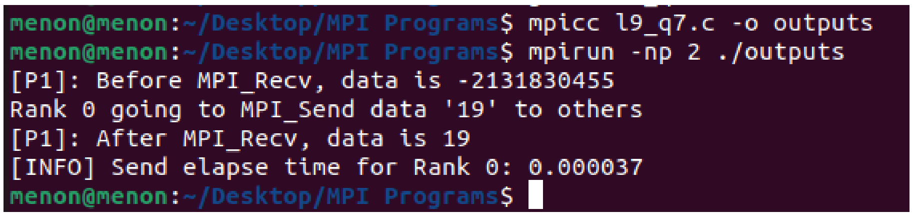
```
menon@menon:~/Desktop/MPI Programs$ mpicc l9_q6.c -o outputs
menon@menon:~/Desktop/MPI Programs$ mpirun -np 2 ./outputs
Rank 0 going to MPI_Isend data '19' to others
[P1]: Before MPI_Irecv, data is -1520545792
[P1]: After MPI_Irecv, data is 19
[INFO] ISend elapse time for Rank 0: 0.000031
menon@menon:~/Desktop/MPI Programs$
```

7. Sample send and receive with mpi_wtime

Code:

```
#include <mpi.h>
#include <stdio.h>

int main(int argc, char** argv)
{
    int rank;
    int buf;
    const int root=0;
    double stime = 0;

    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    int world_size;
    MPI_Comm_size(MPI_COMM_WORLD,&world_size);

    if(rank == root)
    {
        buf = 19;
        printf("Rank %d going to MPI_Send data '%d' to 
others\n",rank,buf);
        stime = - MPI_Wtime();
        for(int i=0;i<world_size;i++){
            MPI_Send(&buf, 1, MPI_INT, i, 0, MPI_COMM_WORLD);
        }
        stime += MPI_Wtime();
    }
    else
    {
        printf("[P%d]: Before MPI_Recv, data is %d\n", rank, buf);
        MPI_Recv(&buf, 1, MPI_INT, 0, 0, MPI_COMM_WORLD,
        MPI_STATUS_IGNORE);
        printf("[P%d]: After MPI_Recv, data is %d\n", rank, buf);
    }
    MPI_Finalize();
    if(rank == root)
    {
        printf("[INFO] Send elapse time for Rank %d: %f\n",rank,stime);
    }
    return 0;
}
```

Output:



```
menon@menon:~/Desktop/MPI Programs$ mpicc l9_q7.c -o outputs
menon@menon:~/Desktop/MPI Programs$ mpirun -np 2 ./outputs
[P1]: Before MPI_Recv, data is -2131830455
Rank 0 going to MPI_Send data '19' to others
[P1]: After MPI_Recv, data is 19
[INFO] Send elapse time for Rank 0: 0.000037
menon@menon:~/Desktop/MPI Programs$ 
```

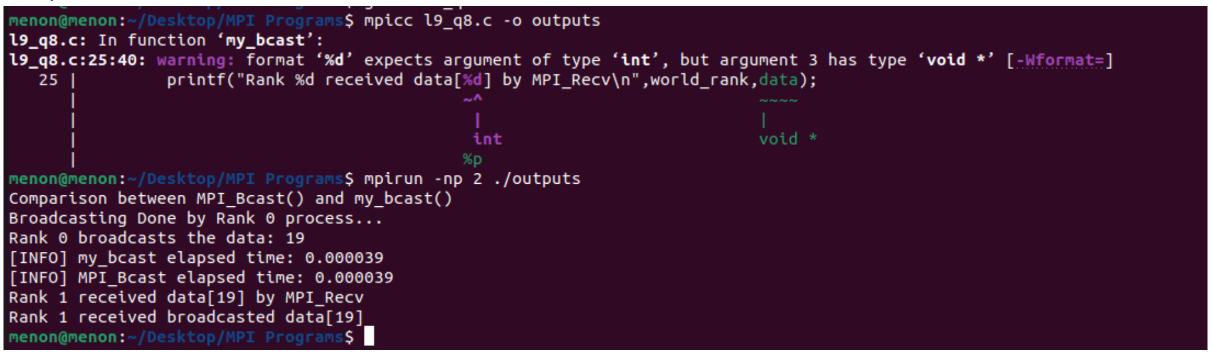
8. Implementing the broadcast using send and receive

Code:

```
#include <mpi.h>
#include <stdio.h>
void my_bcast(void* data, int count, MPI_Datatype datatype, int root,
MPI_Comm communicator)
{
    int world_rank;
    MPI_Comm_rank(communicator, &world_rank);
    int world_size;
    MPI_Comm_size(communicator, &world_size);

    if (world_rank == root)
    {
        // If we are the root process, send our data to everyone
        int i;
        for (i = 0; i < world_size; i++) {
        if (i != world_rank) {
            MPI_Send(data, count, datatype, i, 0, communicator);
        }
        }
    }
    else
    {
        // If we are a receiver process, receive the data from the root
        MPI_Recv(&data, count, datatype, root, 0, communicator,
        MPI_STATUS_IGNORE);
        printf("Rank %d received data[%d] by
MPI_Recv\n",world_rank,data);
    }
}
int main()
{
    MPI_Init(NULL,NULL);
    double total_my_bcast_time = 0;
    double total_mpi_bcast_time = 0;
    int data=19;
    int num_elements = 1;
    int world_rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);

    if(world_rank == 0)
    {
        // Time my_bcast
        total_my_bcast_time -= MPI_Wtime();
        my_bcast(&data, num_elements, MPI_INT, 0, MPI_COMM_WORLD);
        total_my_bcast_time += MPI_Wtime();
```

```c
        printf("Comparison between MPI_Bcast() and 
my_bcast()\nBroadcasting Done by Rank 0 process...\n");
        printf("Rank %d broadcasts the data: %d\n",world_rank,data);
        printf("[INFO] my_bcast elapsed time: 
%f\n",total_my_bcast_time);
        // Time MPI_Bcast
        total_mpi_bcast_time -= MPI_Wtime();
        MPI_Bcast(&data, num_elements, MPI_INT, 0, MPI_COMM_WORLD);
        total_mpi_bcast_time += MPI_Wtime();
        printf("[INFO] MPI_Bcast elapsed time: 
%f\n",total_my_bcast_time);
    }
    else
    {
        my_bcast(&data, num_elements, MPI_INT, 0, MPI_COMM_WORLD);
        MPI_Bcast(&data, num_elements, MPI_INT, 0, MPI_COMM_WORLD);
        printf("Rank %d received broadcasted data[%d]
\n",world_rank,data);
    }
    MPI_Finalize();
    return 0;
}
```

Output:



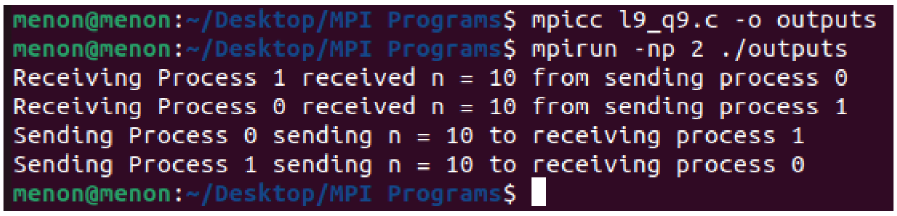
```
menon@menon:~/Desktop/MPI Programs$ mpicc l9_q8.c -o outputs
l9_q8.c: In function ‘my_bcast’:
l9_q8.c:25:40: warning: format ‘%d’ expects argument of type ‘int’, but argument 3 has type ‘void *’ [-Wformat=]
   25 |         printf("Rank %d received data[%d] by MPI_Recv\n",world_rank,data);
      |                                       ~^                             ~~~~
      |                                        |                             |
      |                                        int                           void *
      |                                       %p
menon@menon:~/Desktop/MPI Programs$ mpirun -np 2 ./outputs
Comparison between MPI_Bcast() and my_bcast()
Broadcasting Done by Rank 0 process...
Rank 0 broadcasts the data: 19
[INFO] my_bcast elapsed time: 0.000039
[INFO] MPI_Bcast elapsed time: 0.000039
Rank 1 received data[19] by MPI_Recv
Rank 1 received broadcasted data[19]
menon@menon:~/Desktop/MPI Programs$
```

9. Ring communication

Code:

```c
#include <mpi.h>
#include <stdio.h>

int main(int argc, char *argv[])
{
    int id, p, left, right, n = 10;
    MPI_Request request, request2;
    MPI_Status status;
    MPI_Init(&argc,&argv);
    MPI_Comm_size(MPI_COMM_WORLD, &p);
    MPI_Comm_rank(MPI_COMM_WORLD, &id);

    right = (id + 1) % p;
    left = id - 1;
    if(left < 0) left = p - 1;

    MPI_Isend(&n, 1, MPI_INT, right, 123, MPI_COMM_WORLD, &request2);
    MPI_Wait(&request2, &status);
    MPI_Irecv(&n, 1, MPI_INT, left, 123, MPI_COMM_WORLD, &request);
    MPI_Wait(&request, &status);

    printf("Receiving Process %d received n = %d from sending process %d\n", id, n, left);
    printf("Sending Process %d sending n = %d to receiving process %d\n", id, n, right);
    MPI_Finalize();
    return 0;
}
```

Output:



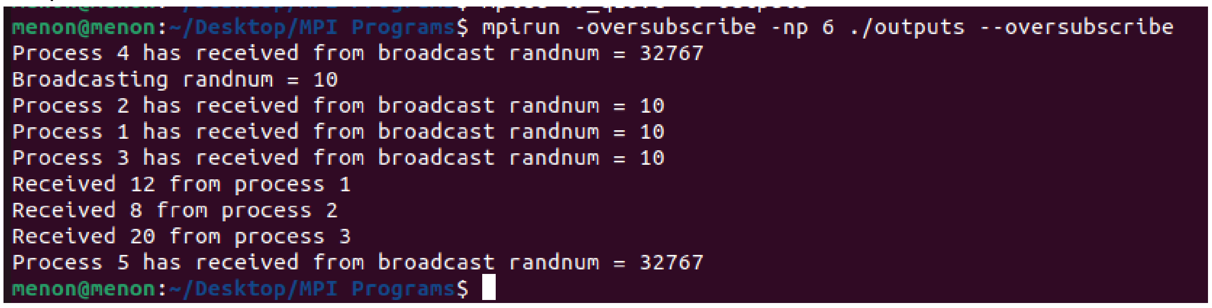
```
menon@menon:~/Desktop/MPI Programs$ mpicc l9_q9.c -o outputs
menon@menon:~/Desktop/MPI Programs$ mpirun -np 2 ./outputs
Receiving Process 1 received n = 10 from sending process 0
Receiving Process 0 received n = 10 from sending process 1
Sending Process 0 sending n = 10 to receiving process 1
Sending Process 1 sending n = 10 to receiving process 0
menon@menon:~/Desktop/MPI Programs$ 
```

10. rank0 - sends randnum, rank1 - Add const , rank2 - sub const, rank3 - mul const

Code:

```c
#include <mpi.h>
#include <stdio.h>

int main(int argc, char** argv) {
    int p, id, randnum, constant = 2;
    MPI_Status status;
    MPI_Init(NULL, NULL);
    MPI_Comm_size(MPI_COMM_WORLD, &p);
    MPI_Comm_rank(MPI_COMM_WORLD, &id);
    if(id == 0)
    {
        randnum = 10;
        printf("Broadcasting randnum = %d\n", randnum);
        MPI_Bcast(&randnum, 1, MPI_INT, 0, MPI_COMM_WORLD);
        int x;
        MPI_Request request1, request2, request3;
        MPI_Irecv(&x, 1, MPI_INT, 1, 111, MPI_COMM_WORLD, &request1);
        MPI_Wait(&request1, &status);
    printf("Received %d from process 1\n", x);
    MPI_Irecv(&x, 1, MPI_INT, 2, 112, MPI_COMM_WORLD, &request2);
        MPI_Wait(&request2, &status);
    printf("Received %d from process 2\n", x);
    MPI_Irecv(&x, 1, MPI_INT, 3, 113, MPI_COMM_WORLD, &request3);
        MPI_Wait(&request3, &status);
    printf("Received %d from process 3\n", x);
    }
    else if(id == 1)
    {
        MPI_Bcast(&randnum, 1, MPI_INT, 0, MPI_COMM_WORLD);
        MPI_Request request;
        int x = randnum + constant;
        MPI_Isend(&x, 1, MPI_INT, 0, 111, MPI_COMM_WORLD, &request);
        MPI_Wait(&request, &status);
    }
    else if(id == 2)
    {
        MPI_Bcast(&randnum, 1, MPI_INT, 0, MPI_COMM_WORLD);
        MPI_Request request;
        int x = randnum - constant;
        MPI_Isend(&x, 1, MPI_INT, 0, 112, MPI_COMM_WORLD, &request);
        MPI_Wait(&request, &status);
    }
    else if(id == 3)
    {
        MPI_Bcast(&randnum, 1, MPI_INT, 0, MPI_COMM_WORLD);
        MPI_Request request;
        int x = randnum * constant;
        MPI_Isend(&x, 1, MPI_INT, 0, 113, MPI_COMM_WORLD, &request);
        MPI_Wait(&request, &status);
    }
    if(id != 0)
    {
        printf("Process %d has received from broadcast randnum = %d\n",
id, randnum);
    }
    MPI_Finalize();
    return 0;
}
```

Output:



```
menon@menon:~/Desktop/MPI Programs$ mpirun -oversubscribe -np 6 ./outputs --oversubscribe
Process 4 has received from broadcast randnum = 32767
Broadcasting randnum = 10
Process 2 has received from broadcast randnum = 10
Process 1 has received from broadcast randnum = 10
Process 3 has received from broadcast randnum = 10
Received 12 from process 1
Received 8 from process 2
Received 20 from process 3
Process 5 has received from broadcast randnum = 32767
menon@menon:~/Desktop/MPI Programs$
```

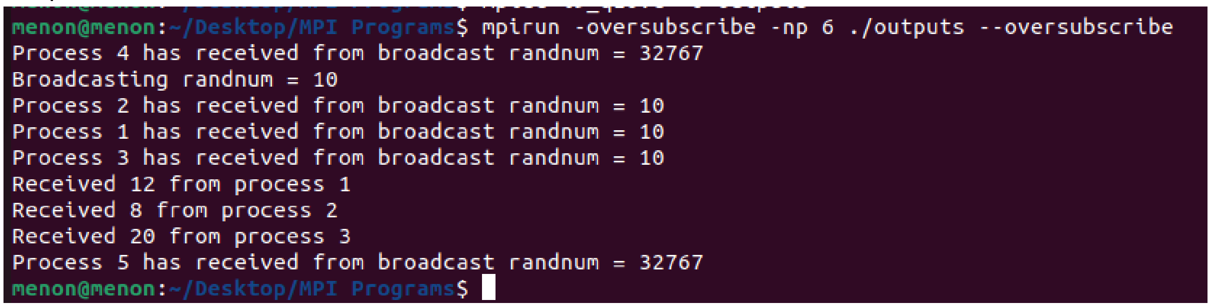
11. Simulate a chat window - server answers query to client

Code:

```
#include <mpi.h>
#include <stdio.h>

int main(int argc, char** argv) {
    int p, id, randnum, constant = 2;
    MPI_Status status;
    MPI_Init(NULL, NULL);
    MPI_Comm_size(MPI_COMM_WORLD, &p);
    MPI_Comm_rank(MPI_COMM_WORLD, &id);
    if(id == 0)
    {
        randnum = 10;
        printf("Broadcasting randnum = %d\n", randnum);
        MPI_Bcast(&randnum, 1, MPI_INT, 0, MPI_COMM_WORLD);
        int x;
        MPI_Request request1, request2, request3;
        MPI_Irecv(&x, 1, MPI_INT, 1, 111, MPI_COMM_WORLD, &request1);
        MPI_Wait(&request1, &status);
    printf("Received %d from process 1\n", x);
    MPI_Irecv(&x, 1, MPI_INT, 2, 112, MPI_COMM_WORLD, &request2);
        MPI_Wait(&request2, &status);
    printf("Received %d from process 2\n", x);
    MPI_Irecv(&x, 1, MPI_INT, 3, 113, MPI_COMM_WORLD, &request3);
        MPI_Wait(&request3, &status);
    printf("Received %d from process 3\n", x);
    }
    else if(id == 1)
    {
        MPI_Bcast(&randnum, 1, MPI_INT, 0, MPI_COMM_WORLD);
        MPI_Request request;
        int x = randnum + constant;
        MPI_Isend(&x, 1, MPI_INT, 0, 111, MPI_COMM_WORLD, &request);
        MPI_Wait(&request, &status);
    }
    else if(id == 2)
    {
        MPI_Bcast(&randnum, 1, MPI_INT, 0, MPI_COMM_WORLD);
        MPI_Request request;
        int x = randnum - constant;
        MPI_Isend(&x, 1, MPI_INT, 0, 112, MPI_COMM_WORLD, &request);
        MPI_Wait(&request, &status);
    }
    else if(id == 3)
    {
        MPI_Bcast(&randnum, 1, MPI_INT, 0, MPI_COMM_WORLD);
```

```c
        MPI_Request request;
        int x = randnum * constant;
        MPI_Isend(&x, 1, MPI_INT, 0, 113, MPI_COMM_WORLD, &request);
        MPI_Wait(&request, &status);
    }
    if(id != 0)
    {
        printf("Process %d has received from broadcast randnum = %d\n",
id, randnum);
    }
    MPI_Finalize();
    return 0;
}
```

Output:



```
menon@menon:~/Desktop/MPI Programs$ mpirun -oversubscribe -np 6 ./outputs --oversubscribe
Process 4 has received from broadcast randnum = 32767
Broadcasting randnum = 10
Process 2 has received from broadcast randnum = 10
Process 1 has received from broadcast randnum = 10
Process 3 has received from broadcast randnum = 10
Received 12 from process 1
Received 8 from process 2
Received 20 from process 3
Process 5 has received from broadcast randnum = 32767
menon@menon:~/Desktop/MPI Programs$
```