Instructions. Write your name in the space above. Fill in the blank to the left of each problem with the correct answer. No points will be subtracted for an incorrect answer, so it is to your advantage to answer all questions.

1. *Mpi* is
   (a) a high-speed interconnect
   (b) a leading supercomputer
   (c) an MPI implementation
   (d) a compute cluster built from off-the-shelf commodity hardware
   (e) an operating system

2. MPI bindings are *not* officially defined for which of the following programming languages:
   (a) C
   (b) C++
   (c) Fortran
   (d) Java

3. MPI stands for
   (a) Multiple Process Interlocutor
   (b) Message Passing Interface
   (c) Multi-Process Interface
   (d) Message-passing Portable Interface
   (e) Message Processing Interlocutor

4. After a successful call of the form `MPI_Wait(&request, &status)` returns, the value stored in `request` will necessarily be
   (a) 0
   (b) `NULL`
   (c) `MPI_REQUEST_NULL`
   (d) `MPI_STATUS_IGNORE`
   (e) the same as it was before the call
5. MPI_Request is
(a) a type
(b) a function
(c) a constant
(d) a variable
(e) a macro

6. MPI_Status is
(a) a type
(b) a function
(c) a constant
(d) a variable
(e) a macro

7. Which of the following is a good reason to use nonblocking MPI communication:
(a) to achieve overlap of computation and computation
(b) to increase the speed of communication
(c) to achieve overlap of computation and communication
(d) to increase the speed of computation
(e) to make a program easier to understand

8. Which of the following is a good reason to use the manager-worker pattern in designing a parallel algorithm:
(a) the number of computational “tasks” is approximately equal to the number of processes
(b) the time required to complete each computational task can be easily predicted
(c) the resulting algorithm will necessarily be deterministic
(d) the time required to complete the computational tasks cannot be easily predicted
(e) to avoid using nondeterministic MPI constructs such as MPI_ANY_SOURCE
Consider the following program:

```c
int main(int argc, char **argv) {
    int rank, nprocs, i, *recvcounts, *displs, total_size, nl, n;
    float *data, *data_l;
    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    nl = rank;
    data_l = (float*)malloc(nl*sizeof(float));
    for (i = 0; i<nl; ++i) data_l[i] = rank;
    total_size = (nprocs*(nprocs-1))/2;
    data = (float*)malloc(total_size*sizeof(float));
    recvcounts = (int*)malloc(nprocs*sizeof(int));
    displs = (int*)malloc(nprocs*sizeof(int));
    for (i=0; i<nprocs; i++) recvcounts[i]=i;
    displs[0]=0;
    for (i=1; i<nprocs; i++) displs[i]=displs[i-1]+recvcounts[i-1];
    MPI_Alltoallv(data_l, nl, MPI_FLOAT, data, recvcounts, displs, MPI_FLOAT, MPI_COMM_WORLD);
    free(data_l); free(displs); free(recvcounts);
    for (n=0; n<nprocs; n++) {
        if (rank == n) {
            printf("Proc %d: ", rank);
            for (i=0; i<total_size; i++) printf("%f ", data[i]);
            printf("\n");
            free(data);
        }
    }
    MPI_Finalize();
    return 0;
}
```

When executed on 4 processes the output is as follows:

```
Proc 0: 1.000000 2.000000 2.000000 3.000000 3.000000 3.000000
Proc 1: 1.000000 2.000000 2.000000 3.000000 3.000000 3.000000
Proc 2: 1.000000 2.000000 2.000000 3.000000 3.000000 3.000000
Proc 3: 1.000000 2.000000 2.000000 3.000000 3.000000 3.000000
```

Which of the following is the missing line:

(a) MPI_Alltoallv(data_l, nl, MPI_FLOAT, data, recvcounts, displs, MPI_FLOAT, MPI_COMM_WORLD)
(b) MPI_Allgatherv(data, recvcounts, nl, MPI_FLOAT, data_l, displs, MPI_FLOAT, MPI_COMM_WORLD)
(c) MPI_Alltoallv(data, recvcounts, nl, MPI_FLOAT, data_l, displs, MPI_FLOAT, MPI_COMM_WORLD)
(d) MPI_Allgatherv(data, nl, displs, recvcounts, MPI_FLOAT, data_l, MPI_FLOAT, MPI_COMM_WORLD)
(e) MPI_Allgatherv(data_l, nl, MPI_FLOAT, data, recvcounts, displs, MPI_FLOAT, MPI_COMM_WORLD)
10. An array of length 10 is stored in the local memory of one MPI process. The programmer wishes to send the first 3 elements of the array to process 0, the next 3 elements to process 1, and the final 4 elements to process 2. Which of the following MPI functions can be used to accomplish this:

(a) MPI_Gather
(b) MPI_Gatherv
(c) MPI_Scatter
(d) MPI_Scatterv
(e) MPI_Bcast
(f) MPI_Bcastv

Each of the following program fragments attempts to have two processes exchange data. In each case, state which of the following is true:

(a) the fragment will definitely deadlock
(b) the fragment will definitely not deadlock
(c) the fragment may or may not deadlock

11. if (rank == 0) {
   MPI_Send(&myNumber, 1, MPI_INT, 1, 9, comm);
   MPI_Recv(&otherNumber, 1, MPI_INT, 1, 9, comm, &status);
} else if (rank == 1) {
   MPI_Send(&myNumber, 1, MPI_INT, 0, 9, comm);
   MPI_Recv(&otherNumber, 1, MPI_INT, 0, 9, comm, &status);
}

12. if (rank == 0) {
   MPI_Recv(&otherNumber, 1, MPI_INT, 1, 9, comm, &status);
   MPI_Send(&myNumber, 1, MPI_INT, 1, 9, comm);
} else if (rank == 1) {
   MPI_Recv(&otherNumber, 1, MPI_INT, 0, 9, comm, &status);
   MPI_Send(&myNumber, 1, MPI_INT, 0, 9, comm);
}

13. if (rank == 0) {
   MPI_Send(&myNumber, 1, MPI_INT, 1, 9, comm);
   MPI_Recv(&otherNumber, 1, MPI_INT, 1, 9, comm, &status);
} else if (rank == 1) {
   MPI_Recv(&otherNumber, 1, MPI_INT, 0, 9, comm, &status);
   MPI_Send(&myNumber, 1, MPI_INT, 0, 9, comm);
}
14. if (rank == 0) {
    MPI_Sendrecv(&myNumber, 1, MPI_INT, 1, 9, &otherNumber, 1, MPI_INT, 1, 9, comm, &status);
} else if (rank == 1) {
    MPI_Sendrecv(&myNumber, 1, MPI_INT, 0, 9, &otherNumber, 1, MPI_INT, 0, 9, comm, &status);
}

15. if (rank == 0) {
    MPI_Irecv(&otherNumber, 1, MPI_INT, 1, 9, comm, &request[0]);
    MPI_Isend(&myNumber, 1, MPI_INT, 1, 9, comm, &request[1]);
} else if (rank == 1) {
    MPI_Irecv(&otherNumber, 1, MPI_INT, 0, 9, comm, &request[0]);
    MPI_Isend(&myNumber, 1, MPI_INT, 0, 9, comm, &request[1]);
} 
MPI_Waitall(2, request, MPI_STATUSES_IGNORE);

16. if (rank == 0) {
    MPI_Isend(&myNumber, 1, MPI_INT, 1, 9, comm, &request[1]);
    MPI_Irecv(&otherNumber, 1, MPI_INT, 1, 9, comm, &request[0]);
} else if (rank == 1) {
    MPI_Isend(&myNumber, 1, MPI_INT, 0, 9, comm, &request[1]);
    MPI_Irecv(&otherNumber, 1, MPI_INT, 0, 9, comm, &request[0]);
} 
MPI_Waitall(2, request, MPI_STATUSES_IGNORE);

17. if (rank == 0) {
    MPI_Send(&myNumber, 1, MPI_INT, 1, 3, comm);
    MPI_Recv(&otherNumber, 1, MPI_INT, 1, 4, comm, &status);
} else if (rank == 1) {
    MPI_Send(&myNumber, 1, MPI_INT, 0, 4, comm);
    MPI_Recv(&otherNumber, 1, MPI_INT, 0, 3, comm, &status);
}

18. In a correct MPI program, the length of the receive buffer used in an MPI_Recv...

(a) must be exactly equal to the length of the incoming message
(b) must be greater than or equal to the length of the incoming message
(c) may be any number; if the length of the buffer is less than the length of the incoming message, the message will be truncated
(d) must be at least one greater than the length of the incoming message
(e) must be less than or equal to the length of the incoming message
19. Which statement best describes what happens whenever the following program is executed with 3 processes:

```c
int main(int argc, char **argv) {
    int rank, x, y;
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    if (rank == 0) {
        x = 10; y = 11;
        MPI_Send(&x, 1, MPI_INT, 1, 9, MPI_COMM_WORLD);
        MPI_Send(&y, 1, MPI_INT, 2, 9, MPI_COMM_WORLD);
    } else if (rank == 1) {
        MPI_Recv(&x, 1, MPI_INT, MPI_ANY_SOURCE, 9, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
        MPI_Recv(&y, 1, MPI_INT, MPI_ANY_SOURCE, 9, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
        printf("%d %d\n", x, y); fflush(stdout);
    } else if (rank == 2) {
        x=20;
        MPI_Recv(&y, 1, MPI_INT, 0, 9, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
        MPI_Send(&x, 1, MPI_INT, 1, 9, MPI_COMM_WORLD);
    }
    MPI_Finalize();
    return 0;
}
```

(a) the program will never deadlock and the output will always be 10 20
(b) the program will never deadlock and the output may be 10 20 or 20 10
(c) it is possible for the program to deadlock, but when it does not deadlock it will output 10 20
(d) the program will always deadlock
(e) it is possible for the program to deadlock, and it is possible for it to output 10 20, and it is possible for it to output 20 10
20. Which statement best describes what will happen whenever the following program is executed with 3 processes:

```c
int main(int argc, char **argv) {
    int rank, x, y;
    MPI_Init( &argc, &argv );
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    if (rank == 0) {
        x = 10; y = 11;
        MPI_Send(&x, 1, MPI_INT, 1, 10, MPI_COMM_WORLD);
        MPI_Send(&y, 1, MPI_INT, 2, 9, MPI_COMM_WORLD);
    } else if (rank == 1) {
        MPI_Recv(&x, 1, MPI_INT, MPI_ANY_SOURCE, 11, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
        MPI_Recv(&y, 1, MPI_INT, MPI_ANY_SOURCE, 10, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
        printf("%d %d\n", x, y); fflush(stdout);
    } else if (rank == 2) {
        x = 20;
        MPI_Recv(&y, 1, MPI_INT, 0, 9, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
        MPI_Send(&x, 1, MPI_INT, 1, 11, MPI_COMM_WORLD);
    }
    MPI_Finalize();
    return 0;
}
```

(a) the program will never deadlock and the output will always be 20 10
(b) the program will never deadlock and the output may be 10 20 or 20 10
(c) it is possible for the program to deadlock, but when it does not deadlock it will output 20 10
(d) the program will always deadlock
(e) it is possible for the program to deadlock, and it is possible for it to output 10 20, and it is possible for it to output 20 10
21. Which statement best describes what will happen whenever the following program is executed with 3 processes:

```c
int main(int argc, char **argv) {
    int rank, x, y;
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    if (rank == 0) {
        x = 10; y = 11;
        MPI_Send(&x, 1, MPI_INT, 1, 10, MPI_COMM_WORLD);
        MPI_Send(&y, 1, MPI_INT, 2, 9, MPI_COMM_WORLD);
    } else if (rank == 1) {
        MPI_Recv(&x, 1, MPI_INT, MPI_ANY_SOURCE, 10, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
        MPI_Recv(&y, 1, MPI_INT, MPI_ANY_SOURCE, 11, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
        printf("%d %d\n", x, y); fflush(stdout);
    } else if (rank == 2) {
        x = 20;
        MPI_Recv(&y, 1, MPI_INT, 0, 9, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
        MPI_Send(&x, 1, MPI_INT, 1, 11, MPI_COMM_WORLD);
    }
    MPI_Finalize();
    return 0;
}
```

(a) the program will never deadlock and the output will always be 10 20
(b) the program will never deadlock and the output may be 10 20 or 20 10
(c) it is possible for the program to deadlock, but when it does not deadlock it will output 10 20
(d) the program will always deadlock
(e) it is possible for the program to deadlock, and it is possible for it to output 10 20, and it is possible for it to output 20 10
22. Which statement best describes what will happen whenever the following program is executed with 2 or more processes:

```c
int main(int argc, char **argv) {
    int rank, x, y;
    MPI_Init( &argc, &argv );
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    if (rank == 0) {
        x = 10; y = 11;
        MPI_Send(&x, 1, MPI_INT, 1, 9, MPI_COMM_WORLD);
        MPI_Send(&y, 1, MPI_INT, 1, 9, MPI_COMM_WORLD);
    } else if (rank == 1) {
        MPI_Recv(&x, 1, MPI_INT, MPI_ANY_SOURCE, 9, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
        MPI_Recv(&y, 1, MPI_INT, MPI_ANY_SOURCE, 9, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
        printf("%d %d\n", x, y); fflush(stdout);
    }
    MPI_Finalize();
    return 0;
}
```

(a) the program will never deadlock and the output will always be 10 11
(b) the program will never deadlock and the output may be 10 11 or 11 10
(c) it is possible for the program to deadlock, but when it does not deadlock it will output 10 11
(d) the program will always deadlock
(e) it is possible for the program to deadlock, and it is possible for it to output 10 11, and it is possible for it to output 11 10

23. Which statement best describes what will happen whenever the following program is executed with 2 or more processes:

```c
int main(int argc, char **argv) {
    int rank, x, y;
    MPI_Init( &argc, &argv );
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    if (rank == 0) {
        x = 10; y = 11;
        MPI_Send(&x, 1, MPI_INT, 1, 15, MPI_COMM_WORLD);
        MPI_Send(&y, 1, MPI_INT, 1, 16, MPI_COMM_WORLD);
    } else if (rank == 1) {
        MPI_Recv(&x, 1, MPI_INT, MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
        MPI_Recv(&y, 1, MPI_INT, MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
        printf("%d %d\n", x, y); fflush(stdout);
    }
    MPI_Finalize();
    return 0;
}
```

(a) the program will never deadlock and the output will always be 10 11
(b) the program will never deadlock and the output may be 10 11 or 11 10
(c) it is possible for the program to deadlock, but when it does not deadlock it will output 10 11
(d) the program will always deadlock
(e) it is possible for the program to deadlock, and it is possible for it to output 10 11, and it is possible for it to output 11 10
24. Which statement best describes what will happen whenever the following program is executed with 2 or more processes:

```c
int main(int argc, char **argv) {
    int rank, x=10, y=20;
    MPI_Request request;
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    if (rank == 0) {
        MPI_Send(&x, 1, MPI_INT, 1, 0, MPI_COMM_WORLD);
        MPI_Isend(&y, 1, MPI_INT, 1, 1, MPI_COMM_WORLD, &request);
        MPI_Wait(&request, MPI_STATUS_IGNORE);
    } else if (rank == 1) {
        MPI_Irecv(&x, 1, MPI_INT, 0, 0, MPI_COMM_WORLD, &request);
        MPI_Recv(&y, 1, MPI_INT, 0, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
        MPI_Wait(&request, MPI_STATUS_IGNORE);
        printf("%d %d\n", x, y); fflush(stdout);
    }
    MPI_Finalize();
    return 0;
}
```

(a) the program will never deadlock and the output will always be 10 20
(b) the program will never deadlock and the output may be 10 20 or 20 10
(c) it is possible for the program to deadlock, but when it does not deadlock it will output 10 20
(d) the program will always deadlock
(e) it is possible for the program to deadlock, and it is possible for it to output 10 20, and it is possible for it to output 20 10

25. Which statement best describes what will happen whenever the following program is executed with 2 or more processes:

```c
int main(int argc, char **argv) {
    int rank, x=10, y=20;
    MPI_Request request;
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    if (rank == 0) {
        MPI_Send(&x, 1, MPI_INT, 1, 0, MPI_COMM_WORLD);
        MPI_Isend(&y, 1, MPI_INT, 1, 1, MPI_COMM_WORLD, &request);
        MPI_Wait(&request, MPI_STATUS_IGNORE);
    } else if (rank == 1) {
        MPI_Irecv(&x, 1, MPI_INT, 0, 0, MPI_COMM_WORLD, &request);
        MPI_Recv(&y, 1, MPI_INT, 0, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
        MPI_Wait(&request, MPI_STATUS_IGNORE);
        printf("%d %d\n", x, y); fflush(stdout);
    }
    MPI_Finalize();
    return 0;
}
```

(a) the program will never deadlock and the output will always be 20 10
(b) the program will never deadlock and the output may be 10 20 or 20 10
(c) it is possible for the program to deadlock, but when it does not deadlock it will output 20 10
(d) the program will always deadlock
(e) it is possible for the program to deadlock, and it is possible for it to output 10 20, and it is possible for it to output 20 10

For each of the following, answer True (T) or False (F):

_____ 26. A correct MPI program may safely read from the send buffer used in a nonblocking send operation, after the send has been posted but before it has been completed by a call to MPI_Wait.

_____ 27. A correct MPI program may safely write to the send buffer used in a nonblocking send operation, after the send has been posted but before it has been completed by a call to MPI_Wait.

_____ 28. A correct MPI program may safely read from the receive buffer used in a nonblocking receive operation, after the receive has been posted but before it has been completed by a call to MPI_Wait.

_____ 29. A correct MPI program may safely write to the receive buffer used in a nonblocking receive operation, after the receive has been posted but before it has been completed by a call to MPI_Wait.

_____ 30. A correct MPI program may safely post two send requests using the same send buffer before waiting on either request.

_____ 31. A correct MPI program may safely post a send request, wait on that request, then post a second send request using the same send buffer as the first, and then wait on the second request.

_____ 32. A program contains a call to MPI_Bcast with data type MPI_DOUBLE used on every process. Which of the following must be true if the program is correct:

(a) the count argument used on a non-root process must be exactly equal to the count on the root
(b) the count arguments used on non-root processes can differ, as long as they are all greater than or equal to the count on the root
(c) the count on all the non-root processes must be the same number, but that number may be larger than the count used on the root
(d) the count on all the non-root processes must be the same number, but that number may be at least one larger than the count used on the root
(e) the count values on the non-root processes can be any numbers; if they are smaller than the count on the root, the message will just be truncated.
33. Which statement is true of the following program:

```c
#include<stdio.h>
#include<mpi.h>
int main(int argc, char **argv) {
    int rank, x=1;
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    if (rank == 0) {
        MPI_Send(&x, 1, MPI_INT, 1, 9, MPI_COMM_WORLD);
    } else if (rank == 1) {
        MPI_Recv(&x, 1, MPI_INT, MPI_ANY_SOURCE, 9, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
        printf("Message received.\n");
    }
    return 0;
}
```

(a) When run with 2 or more processes, the program will never deadlock and will output “Message received.”.
(b) When run with more than 2 processes, the program may or may not deadlock; if it does not deadlock, it will output “Message received.”.
(c) When run with more than 2 processes, the program will deadlock.
(d) When run with 1 process, the program will always terminate normally without printing anything.
(e) The program is incorrect.

34. How many primes are there between 1 and 50 (inclusive)?

35. A Beowulf system is

(a) a high-performance disc array
(b) a specialized multiprocessor computer with a high-speed interconnect, such as the IBM BG/P
(c) a compute cluster built from commodity off-the-shelf components
(d) a large-scale shared memory supercomputer, or
(e) an architecture based on a single Very High Frequency Core (VHFC) CPU.

36. Consider the `diffusion1d` program developed in class. Suppose that the length of the global temperature array is 100, and the program is executed with 10 processes. What is the maximum number of ghost cells stored on any one process?

37. Consider the row-distributed `diffusion2d` program developed in the homework assignment. Suppose that the dimensions of the global temperature matrix is $100 \times 100$, and the program is executed with 10 processes. What is the maximum number of ghost cells stored on any one process?
For the following, suppose an array of length $n$ (indexed from 0 to $n - 1$) is block-distributed over $p$ processes (with ranks $0, \ldots, p - 1$).

38. What is the formula $\text{first}(i)$ for the global index of the first element on process $i$?

39. What is the formula $\text{num}(i)$ for the number of elements stored on the process of rank $i$?

40. What is the formula for the rank $i$ of the process controlling the element with global index $j$?

41. What is the formula for the local index $k$ of the element with global index $j$?

42. What is the formula for the global index $j$ of the element with local index $k$ on the process of rank $i$?

For the following questions, suppose $n = 1374$ and $p = 16$.

43. What is the number of elements stored on the process of rank 6?

44. What is the rank of the process controlling the element with global index 937?

45. What is the local index of the element with global index 937?

46. What is the global index of the element with local index 27 on the process of rank 7?
47. Suppose every process in a communicator calls `MPI_Reduce` (correctly) with `MPI_SUM` as the reduction operation. Does this necessarily induce a barrier? (Y/N)

48. Suppose every process in a communicator calls `MPI_Allreduce` (correctly) with `MPI_SUM` as the reduction operation. Does this necessarily induce a barrier? (Y/N)

49. Suppose every process in a communicator calls `MPI_Bcast` (correctly). Does this necessarily induce a barrier? (Y/N)

50. Which statement best describes what will happen whenever the following program is executed with 3 processes:

```c
int main(int argc, char **argv) {
    int rank, x=10, y=20, u=30, v;
    MPI_Init( &argc, &argv );
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    if (rank == 0) {
        MPI_Recv(&x, 1, MPI_INT, MPI_ANY_SOURCE, 9, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
        MPI_Reduce(&u, &v, 1, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD);
        MPI_Recv(&y, 1, MPI_INT, MPI_ANY_SOURCE, 9, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
        printf("%d %d\n", x, y); flush(stdout);
    } else if (rank == 1) {
        MPI_Send(&x, 1, MPI_INT, 0, 9, MPI_COMM_WORLD);
        MPI_Reduce(&u, &v, 1, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD);
    } else if (rank == 2) {
        MPI_Reduce(&u, &v, 1, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD);
        MPI_Send(&y, 1, MPI_INT, 0, 9, MPI_COMM_WORLD);
    }
    MPI_Finalize();
    return 0;
}
```

(a) the program will never deadlock and the output will always be 10 20
(b) the program will never deadlock and the output may be 10 20 or 20 10
(c) it is possible for the program to deadlock, but when it does not deadlock it will output 10 20
(d) the program will always deadlock
(e) it is possible for the program to deadlock, and it is possible for it to output 10 20, and it is possible for it to output 20 10