The following C program sums up all the values in array "data" and displays the sum total:

```
 ~
#include <stdio.h>
#define NUMDATA 10000
int data[NUMDATA];
void LoadData(int data[])
{
    for(int i = 0; i < NUMDATA; i++){
        data[i] = 1;
    }
}
int AddUp(int data[], int count)
{
    int sum = 0;
    for(int i = 0; i < count; i++){
        sum += data[i];
    }
    return sum;
}
int main(void) {
    int sum;
    LoadData(data);
    sum = AddUp(data, NUMDATA);
    printf("The total sum of data is %d\n", sum);
    return 0;
}
```

MPI-ed

```
 ~

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

#define NUMDATA 10000


int main(int argc, char *argv[]) {
    int rank, size;
    int local_sum = 0, global_sum = 0;
    int local_data[NUMDATA];

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

    int chunk_size = NUMDATA / size;
    for (int i = 0; i < chunk_size; i++) {
        local_data[i] = 1;
    }

    for (int i = 0; i < chunk_size; i++) {
        local_sum += local_data[i];
    }

    MPI_Reduce(&local_sum, &global_sum, 1, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD);

    if (rank == 0) {
        printf("The total sum of data is %d\n", global_sum);
    }

    MPI_Finalize();
    return 0;
}
```

```
 ~

[wizard@archlinux code]$ mpirun -np 4 ./a.out
The total sum of data is 10000
[wizard@archlinux code]$
```

Write an MPI program called pingpong.c to run with exactly 2 processes. Process rank 0 is to send an integer variable called "ball" initialised with the value zero to Process rank 1. Process rank 1 will add 1 to the ball and send it back. This will repeat until the ball has a value of 10 in Process rank 0.

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

int main(int argc, char *argv[]) {
    int rank, received = 0;
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    if (size != 2) {
        if (rank == 0) {
            printf("Error: This program requires exactly 2 processes.\n");
        }
        MPI_Finalize();
        return 1;
    }

    if (rank == 0) {
        for (int i = 0; i < 10; i++) {
            printf("Process 0 sent: %d", received);
            MPI_Send(&received, 1, MPI_INT, 1, 0, MPI_COMM_WORLD);
            MPI_Recv(&received, 1, MPI_INT, 1, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
            printf("; Process 0 received: %d\n", received);
        }
    } else if (rank == 1) {
        for (int i = 0; i < 10; i++) {
            MPI_Recv(&received, 1, MPI_INT, 0, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
            printf("Process 1 received: %d", received);
            received++;
            MPI_Send(&received, 1, MPI_INT, 0, 0, MPI_COMM_WORLD);
            printf("; Process 1 sent: %d\n", received);
        }
    }

    MPI_Finalize();
    return 0;
}
```

```
~

Process 0 sent: 0; Process 0 received: 1
Process 0 sent: 1; Process 0 received: 2
Process 0 sent: 2; Process 0 received: 3
Process 0 sent: 3; Process 0 received: 4
Process 0 sent: 4; Process 0 received: 5
Process 0 sent: 5; Process 0 received: 6
Process 0 sent: 6; Process 0 received: 7
Process 0 sent: 7; Process 0 received: 8
Process 0 sent: 8; Process 0 received: 9
Process 0 sent: 9; Process 0 received: 10
Process 1 received: 0; Process 1 sent: 1
Process 1 received: 1; Process 1 sent: 2
Process 1 received: 2; Process 1 sent: 3
Process 1 received: 3; Process 1 sent: 4
Process 1 received: 4; Process 1 sent: 5
Process 1 received: 5; Process 1 sent: 6
Process 1 received: 6; Process 1 sent: 7
Process 1 received: 7; Process 1 sent: 8
Process 1 received: 8; Process 1 sent: 9
Process 1 received: 9; Process 1 sent: 10
```

Write a "Pass-the-parcel" MPI program that will run with 3 or more nodes, such that Process rank 0 will send an integer variable call "parcel" initialised with 1, to Process rank 1 which will add 1 to the parcel and then send it to Process rank 2, and so on until the highest rank process will send it back to Process rank 0, at which point the parcel variable should contain the value of the number of nodes there are.

```
~

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

int main(int argc, char *argv[]) {
    int rank, size, received = 1;

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

    if (size <= 3) {
        if (rank == 0) {
            printf("Error: This program requires exactly 2 processes.\n");
        }
        MPI_Finalize();
        return 1;
    }

    if (rank == 0) {
        MPI_Send(&received, 1, MPI_INT, 1, 0, MPI_COMM_WORLD);
        MPI_Recv(&received, 1, MPI_INT, size - 1, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
        printf("Finalized, and got back: %d\n", received);
    } else {
        MPI_Recv(&received, 1, MPI_INT, rank - 1, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
        received++;

        if (rank < size - 1) {
            MPI_Send(&received, 1, MPI_INT, rank + 1, 0, MPI_COMM_WORLD);
        } else {
            MPI_Send(&received, 1, MPI_INT, 0, 0, MPI_COMM_WORLD);
        }
    }

    MPI_Finalize();
    return 0;
}
```

```
~

[wizard@archlinux code]$ mpirun -np 4 ./a.out
Finalized, and got back: 4
```

The following program has all the processes with rank greater than 0 send random amounts of data to Process rank 0 :

~

```c
#include <mpi.h>
#include <stdio.h>
#include <stdlib.h>
#define NUMDATA 1000
int main(int argc, char **argv)
{
    int size;
    int rank;
    int tag=0;
    int count;
    MPI_Status status;
    int data[NUMDATA];
    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    if (rank == 0){
        for(int i = 0; i < size - 1; i++) {
            MPI_Recv(data, NUMDATA, MPI_INT, MPI_ANY_SOURCE, MPI_ANY_TAG,
                    MPI_COMM_WORLD, &status);
            MPI_Get_count(&status, MPI_INT, &count);
            printf("Node ID: %d; tag: %d; MPI_Get_count: %d; \n",
                    status.MPI_SOURCE, status.MPI_TAG, count);
        }
    }
    else {
        MPI_Send(data, rand()%100, MPI_INT, 0, tag, MPI_COMM_WORLD);
    }
    MPI_Finalize();
    return 0;
}
```

Dynamic-ed

```
 ~

#include <mpi.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main(int argc, char **argv) {
    int size, rank, tag = 0, count;
    MPI_Status status;

    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    srand(time(NULL) + rank);
    if (rank == 0) {
        for (int i = 0; i < size - 1; i++) {

            // Probe allows to see who sent; without actually like receiving it while
still allowing the status
            MPI_Probe(MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD, &status);
            MPI_Get_count(&status, MPI_INT, &count);

            int *data = (int *)malloc(count * sizeof(int));
            MPI_Recv(data, count, MPI_INT, status.MPI_SOURCE, status.MPI_TAG,
MPI_COMM_WORLD, MPI_STATUS_IGNORE);
            printf("Node ID: %d; tag: %d; Received count: %d\n", status.MPI_SOURCE,
status.MPI_TAG, count);

            free(data);
        }
    } else {
        count = rand() % 100;
        int *data = (int *)malloc(count * sizeof(int));
        MPI_Send(data, count, MPI_INT, 0, tag, MPI_COMM_WORLD);
        free(data);
    }

    MPI_Finalize();
    return 0;
}
```

```
 ~

[wizard@archlinux code]$ mpirun -np 4 ./a.out
Node ID: 1; tag: 0; Received count: 62
Node ID: 2; tag: 0; Received count: 89
Node ID: 3; tag: 0; Received count: 43
```

WarAndPeace.txt letter count:

~

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

void count_letters_in_chunk(char *chunk, int chunk_size, int *local_counts) {
    for (int i = 0; i < chunk_size; i++) {
        char c = tolower(chunk[i]);
        if (c >= 'a' && c <= 'z') {
            local_counts[c - 'a']++;
        }
    }
}

int main(int argc, char *argv[]) {
    int rank, size;
    char *filename = "../WarAndPeace.txt";

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

    int global_counts[26] = {0};
    int local_counts[26] = {0};

    if (rank == 0) {
        FILE *file = fopen(filename, "r");
        if (!file) {
            printf("Failed to open file %s\n", filename);
            MPI_Abort(MPI_COMM_WORLD, 1);
        }

        fseek(file, 0, SEEK_END);
        long file_size = ftell(file);
        fseek(file, 0, SEEK_SET);

        long chunk_size = file_size / size;
        long remaining = file_size % size;

        char *buffer = (char *)malloc(chunk_size + 1);

        for (int i = 1; i < size; i++) {
            long offset = i * chunk_size;
            if (i == size - 1) {
                chunk_size += remaining;
            }

            fseek(file, offset, SEEK_SET);
            fread(buffer, 1, chunk_size, file);

            MPI_Send(&chunk_size, 1, MPI_LONG, i, 0, MPI_COMM_WORLD);
            MPI_Send(buffer, chunk_size, MPI_CHAR, i, 0, MPI_COMM_WORLD);
        }

        fread(buffer, 1, chunk_size, file);
        count_letters_in_chunk(buffer, chunk_size, local_counts);

        fclose(file);
        free(buffer);
    } else {

        long chunk_size;
```

```
        MPI_Recv(&chunk_size, 1, MPI_LONG, 0, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);

        char *buffer = (char *)malloc(chunk_size + 1);
        MPI_Recv(buffer, chunk_size, MPI_CHAR, 0, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);

        count_letters_in_chunk(buffer, chunk_size, local_counts);
        free(buffer);
    }

    MPI_Reduce(local_counts, global_counts, 26, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD);

    if (rank == 0) {
        printf("Letter counts in 'War and Peace':\n");
        for (int i = 0; i < 26; i++) {
            printf("%c: %d\n", 'a' + i, global_counts[i]);
        }
    }

    MPI_Finalize();
    return 0;
}
```

```
 ~

a: 202110
b: 34208
c: 62019
d: 117862
e: 314836
f: 56371
g: 49360
h: 168087
i: 170975
j: 2441
k: 18996
l: 95572
m: 61535
n: 183713
o: 189951
p: 45373
q: 2343
r: 147007
s: 164733
t: 227912
u: 63350
v: 26487
w: 58939
x: 4205
y: 45833
z: 2351
```

The sum of those numbers : 2,502,497

```
 ~ Verification: This just gives the total number of letteres in the text file, capitals are
treated as small; non alphabeticals are ignored

[wizard@archlinux code]$ tr -cd '[:alpha:]' < ../WarAndPeace.txt | wc -m
2502497
[wizard@archlinux code]$
```