

|  |  |
| --- | --- |
| **ASSIGNMENT** | |
| **Course Code** | CSC303A |
| **Course Name** | Computer Networks |
| **Programme** | B. Tech. |
| **Department** | Computer Science and Engineering |
| **Faculty** | Faculty of Engineering & Technology |

#### 

|  |  |
| --- | --- |
| **Name of the Student** | Subhendu Maji |
| **Reg. No** | 18ETCS002121 |
| **Semester/Year** | 5TH semester / 2018 batch |
| **Course Leader/s** | Mr. Nithin Rao R |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Declaration Sheet** | | | | | | | | |
| Student Name | Subhendu Maji | | | | | | | |
| Reg. No | 18ETCS002121 | | | | | | | |
| Programme | B. Tech. | | | | | Semester/Year | 5th sem / 2018 batch | |
| Course Code | CSC303A | | | | | | | |
| Course Title | Computer Networks | | | | | | | |
| Course Date |  | | to | |  | | | |
| Course Leader | Mr. Nithin Rao R | | | | | | | |
| **Declaration**  The assignment submitted herewith is a result of my own investigations and that I have conformed to the guidelines against plagiarism as laid out in the Student Handbook. All sections of the text and results, which have been obtained from other sources, are fully referenced. I understand that cheating and plagiarism constitute a breach of University regulations and will be dealt with accordingly. | | | | | | | | |
| Signature of the Student | |  | | | | | Date |  |
| Submission date stamp  (by Examination & Assessment Section) | |  | | | | | | |
| Signature of the Course Leader and date | | | | Signature of the Reviewer and date | | | | |
|  | | | |  | | | | |

# **Contents**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

[**Declaration Sheet** ii](#_Toc58039117)

[**Contents** iii](#_Toc58039118)

[**Marking Scheme** iv](#_Toc58039119)

[**Question No. 1** 5](#_Toc58039120)

[1.1 Disadvantages of the protocol 7](#_Toc58039121)

[1.2 Modifications to overcome the disadvantages 9](#_Toc58039122)

[**Question No. 2** 11](#_Toc58039123)

[2.1 Program to compute checksum at the transmitter 11](#_Toc58039124)

[2.2 Program to check for error free data transmission at the receiver 16](#_Toc58039125)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Assignment - 1** | | | |  |  |  |
| Register No. | | **18ETCS002121** | Name of Student |  | **SUBHENDU MAJI** |  |
| **Sections** |  | **Marking Scheme** | | **Max Marks** | **First Examiner**  **Marks** | **Second Examiner**  **M**  **arks** |
| **Q.1** | 1.1 | Disadvantages of the protocol | | 02 |  |  |
| 1.2 | Modifications to overcome the disadvantages | | 03 |  |  |
|  | **Max Marks** | | **05** |  |  |
| **Q.2** | 2.1 | Program to compute checksum at the transmitter | | 10 |  |  |
| 2.2 | Program to check for error free data transmission at the receiver | | 10 |  |  |
|  | **Max Marks** | | **20** |  |  |
|  | **Total Assignment Marks** | | | **25** |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Course Marks Tabulation** | | | | |
| **Component- 1(B)**  **Assignment** | **First**  **Examiner** | **Remarks** | **Second Examiner** | **Remarks** |
| Q 1 |  |  |  |  |
| Q 2 |  |  |  |  |
| **Marks (Max 25 )** |  |  |  |  |
| Signature of First Examiner Signature of Second Examiner | | | | |

# **Question No. 1**

**Solution to Question No. 1:**

Stop and Wait Protocol is a data-link layer protocol which is used for transmitting the data over the noiseless channels. It provides unidirectional data transmission which means that either sending or receiving of data will take place at a time. It provides flow-control mechanism but does not provide any error control mechanism.

The idea behind the usage of this frame is that when the sender sends the frame then he waits for the acknowledgment before sending the next frame.

The primitives of stop and wait protocol are:

**Sender side**

Rule 1: Sender sends one data packet at a time.

Rule 2: Sender sends the next packet only when it receives the acknowledgment of the previous packet.

Therefore, the idea of stop and wait protocol in the sender's side is very simple, i.e., send one packet at a time, and do not send another packet before receiving the acknowledgment.

**Receiver side**

Rule 1: Receive and then consume the data packet.

Rule 2: When the data packet is consumed, receiver sends the acknowledgment to the sender.

Therefore, the idea of stop and wait protocol in the receiver's side is also very simple, i.e., consume the packet, and once the packet is consumed, the acknowledgment is sent. This is known as a flow control mechanism.

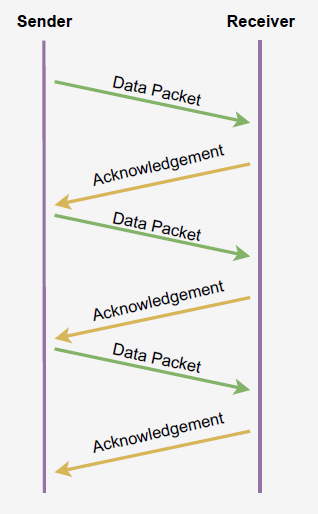


Figure stop-and-wait protocol

If there is a sender and receiver, then sender sends the packet and that packet is known as a data packet. The sender will not send the second packet without receiving the acknowledgment of the first packet. The receiver sends the acknowledgment for the data packet that it has received. Once the acknowledgment is received, the sender sends the next packet. This process continues until all the packet are not sent. The main advantage of this protocol is its simplicity but it has some disadvantages also. For example, if there are 1000 data packets to be sent, then all the 1000 packets cannot be sent at a time as in Stop and Wait protocol, one packet is sent at a time.

## Disadvantages of the protocol

1. Lost Data

Suppose the sender sends the data and the data is lost. The receiver is waiting for the data for a long time. Since the data is not received by the receiver, so it does not send any acknowledgment. Since the sender does not receive any acknowledgment so it will not send the next packet. This problem occurs due to the lost data.

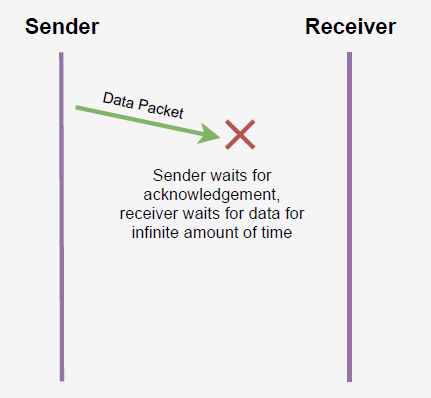


Figure Data Lost

In this case, two problems occur:

* Sender waits for an infinite amount of time for an acknowledgment.
* Receiver waits for an infinite amount of time for a data.

1. Lost Acknowledgment

Suppose the sender sends the data and it has also been received by the receiver. On receiving the packet, the receiver sends the acknowledgment. In this case, the acknowledgment is lost in a network, so there is no chance for the sender to receive the acknowledgment. There is also no chance for the sender to send the next packet as in stop and wait protocol, the next packet cannot be sent until the acknowledgment of the previous packet is received.

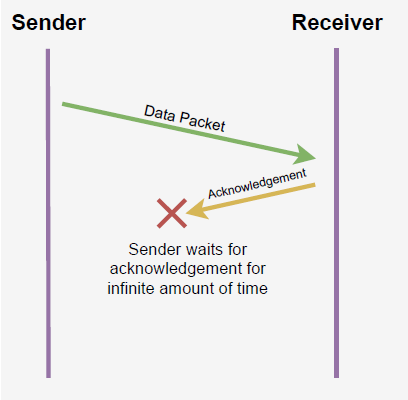


Figure Acknowledgement Lost

In this case, one problem occurs:

* Sender waits for an infinite amount of time for an acknowledgment.

1. Delayed Data or Acknowledgment

Suppose the sender sends the data and it has also been received by the receiver. The receiver then sends the acknowledgment but the acknowledgment is received after the timeout period on the sender's side. As the acknowledgment is received late, so acknowledgment can be wrongly considered as the acknowledgment of some other data packet.

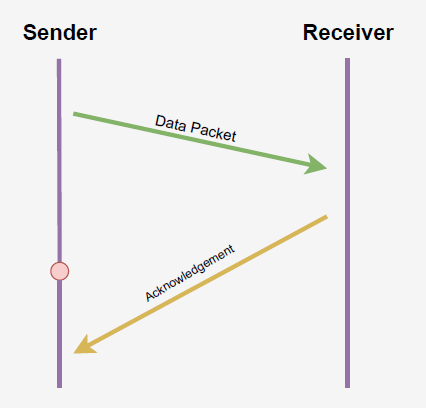


Figure delayed data or acknowledgment

## 1.2 Modifications to overcome the disadvantages

Above 3 problems are resolved by Stop and Wait ARQ (Automatic Repeat Request) that does both error control and flow control.

1. **Time Out**

The problems due to lost data can be solved by using a time-out timer, after transmitting the data packet to the receiver through the communication channel, the sender starts the time out timer. Now if the data packet's feedback is received by the sender before the timer expires, then the sender stops the timer and transmits the next data packet.

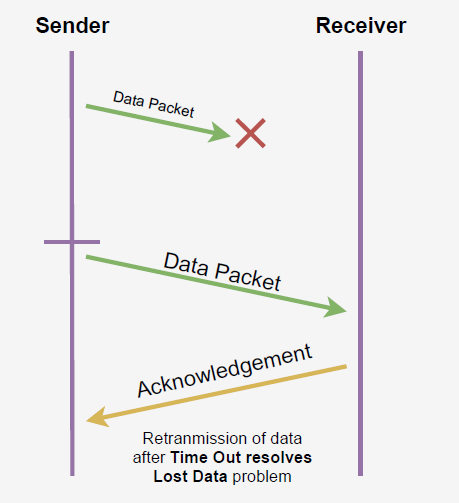


Figure Time out to solve data lost problem

1. **Sequence Number** (Data)

The problem of lost acknowledgement is solved by putting sequence numbers on data packets. Assume that the feedback given by the receiver is lost, then the sender resends the same data packet after its timer expires. This prevents the occurrence of deadlock. The sequence number on the data packets helps the receiver to identify the duplicate data packet, then the receiver discards the duplicate data packet & resends the same acknowledgement.

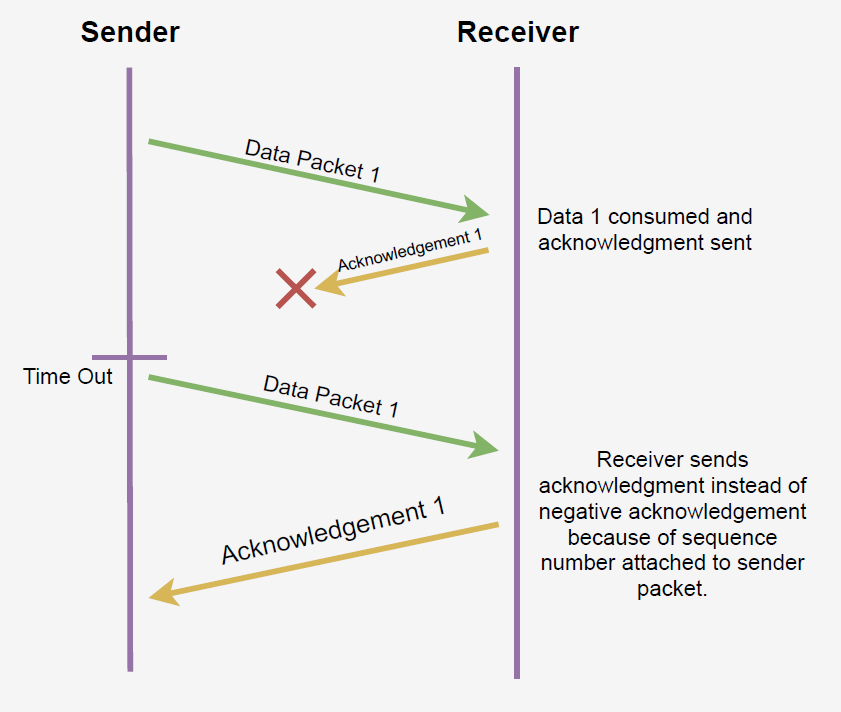


Figure sequence number with data to solve acknowledgement lost problem

1. **Delayed Acknowledgement**

The problem of delayed feedback is solved by putting sequence numbers on the acknowledgement. Assume that the feedback given by the receiver is lost, then the sender resends the same data packet after its timer expires. The sequence number on the data packets helps the receiver to identify the duplicate data packet, then the receiver discards the duplicate data packet & resends the same feedback. Meanwhile if the delayed feedback also reaches the sender, the sender discards the duplicate feedback based on the feedback sequence number.

This is resolved by introducing sequence number for acknowledgement also.

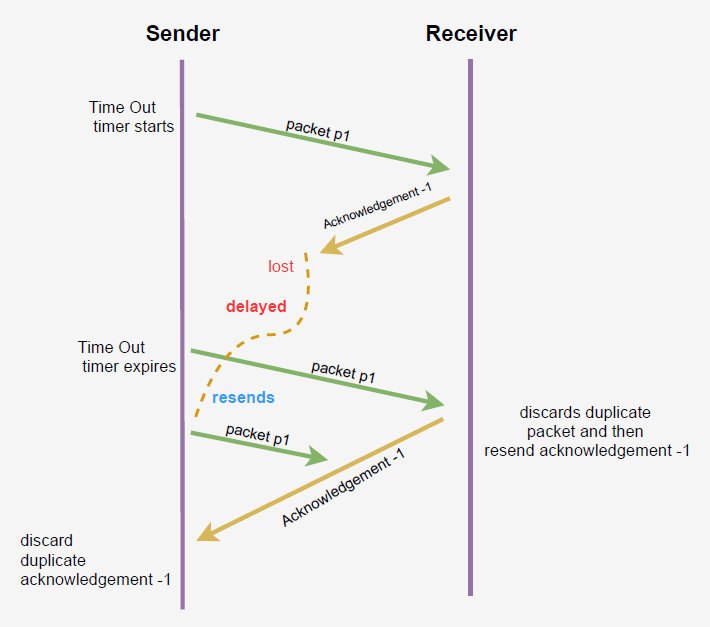


Figure sequence number with acknowledgement to solve delayed data/acknowledgemnt problem

# **Question No. 2**

**Solution to Question No. 2:**

## 2.1 Program to compute checksum at the transmitter

Algorithm to compute checksum at the transmitter

STEP 1: Start

STEP 2: Input data

STEP 3: divide data into n blocks with k elements each and store each block in an array

STEP 4: do binary addition of all the blocks and store in sum array

STEP 5: if the length of the sum is more than k, again do the binary addition of the carry and the sum

STEP 6: complement the sum to get the checksum

STEP 7: Print checksum

STEP 8: Stop

Program to compute checksum at the transmitter

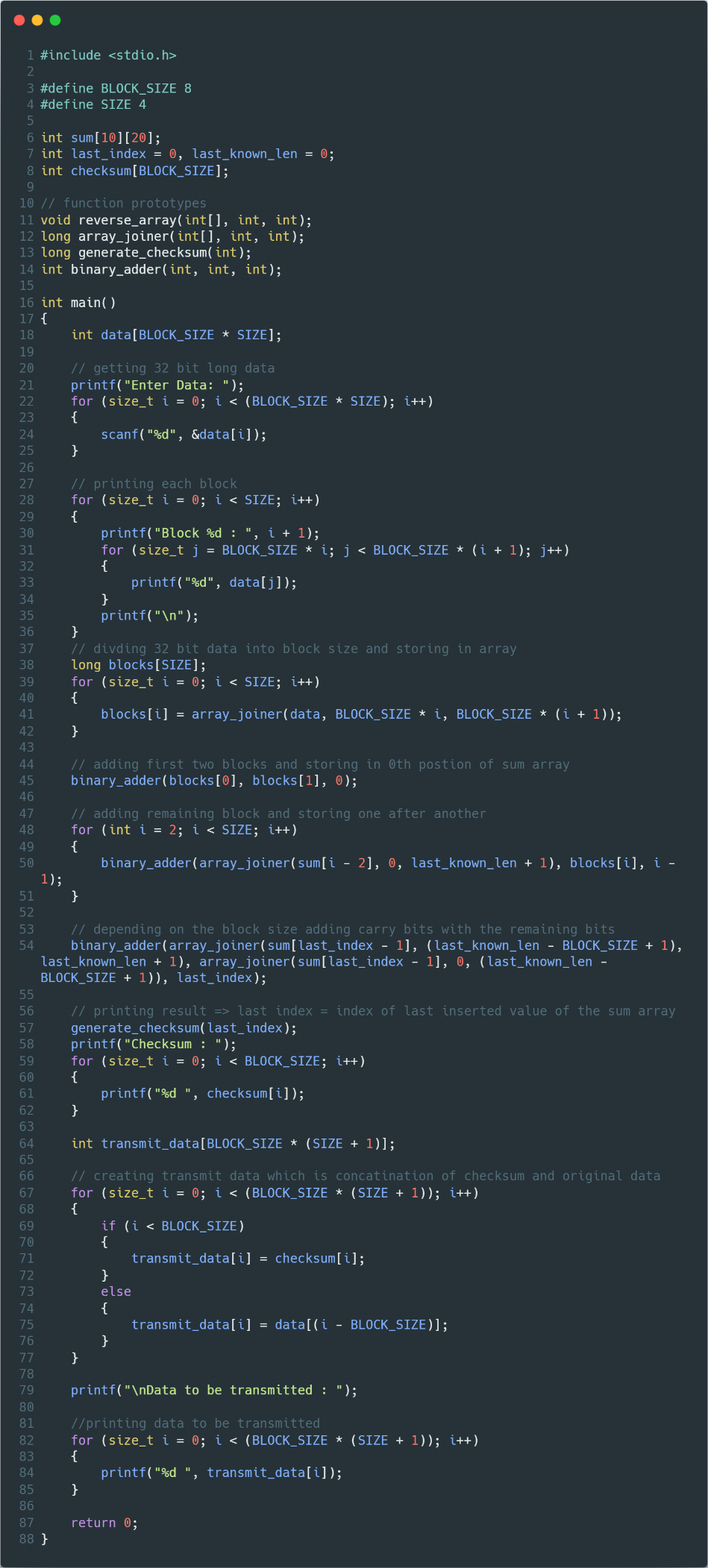


Figure Driver Program

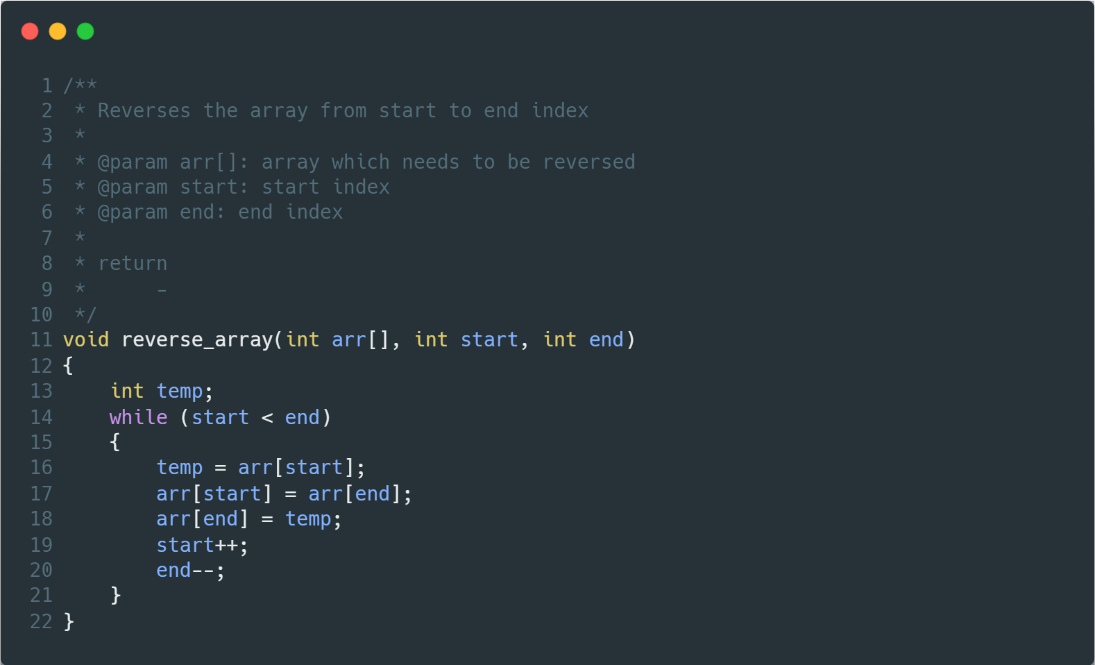


Figure Function to reverse an array



Figure function to join array and return a number



Figure Function to generate checksum



Figure Function to add two binary numbers

Output

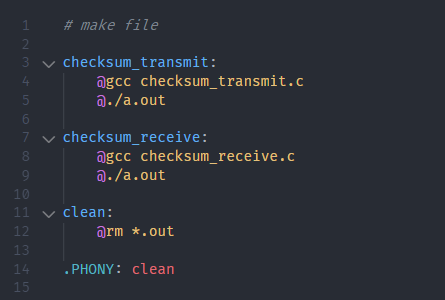


Figure makefile

Let’s say we want to send a -bit long data:

It can be divided into blocks.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Block 1 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 |
| Block 2 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| Block 3 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 |
| Block 4 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 |

Adding all the block we get

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Carry bits | 1 | 0  1 | 0 | 1 | 1 | 1 | 1 | 1 |  |  |
| Block 1 |  |  | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 |
| Block 2 |  |  | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| Block 3 |  |  | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 |
| Block 4 |  | **+** | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 |
|  |  |  |  |  |  |  |  |  |  |  |
| Sum | **1** | **1** | **0** | **0** | **1** | **0** | **1** | **0** | **0** | **1** |

Adding carry bits with the sum,

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Sum |  |  | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 |
| Carry |  |  |  |  |  |  |  | **+** | 1 | 1 |
|  |  |  |  |  |  |  |  |  |  |  |
| Final Sum |  |  | **0** | **0** | **1** | **0** | **1** | **1** | **0** | **0** |
| checksum |  |  | **1** | **1** | **0** | **1** | **0** | **0** | **1** | **1** |

Data to be transmitted becomes:

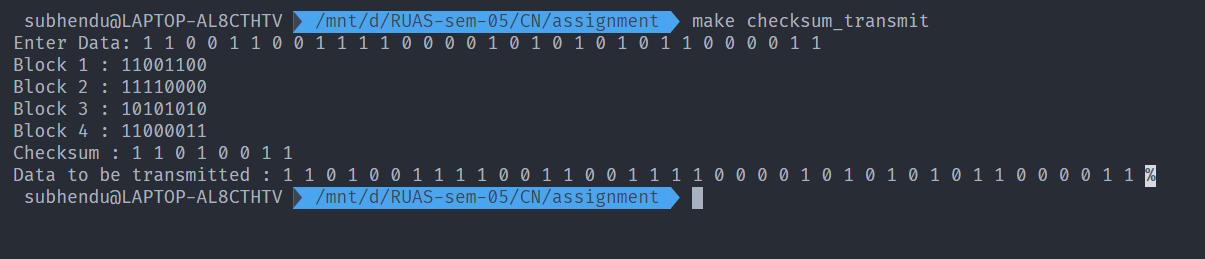


Figure execution of checksum\_transmit

## 2.2 Program to check for error free data transmission at the receiver

Algorithm to check for error free transmission at the receiver

STEP 1: Start

STEP 2: Input received data

STEP 3: divide data into n blocks with k elements each and store each block in an array

STEP 4: First k elements or the first block is the checksum

STEP 4: do binary addition of all the blocks and the checksum and store in sum array

STEP 5: if the length of the sum is more than k, again do the binary addition of the carry and the sum

STEP 6: if the final sum contains all 1’s, then the received data is error free, else the received data is **not** error free.

STEP 7: print the sum

STEP 8: print ‘ERROR’ if the sum contains 0, else print ‘NO ERROR’

STEP 9: Stop

Program to compute checksum at the transmitter

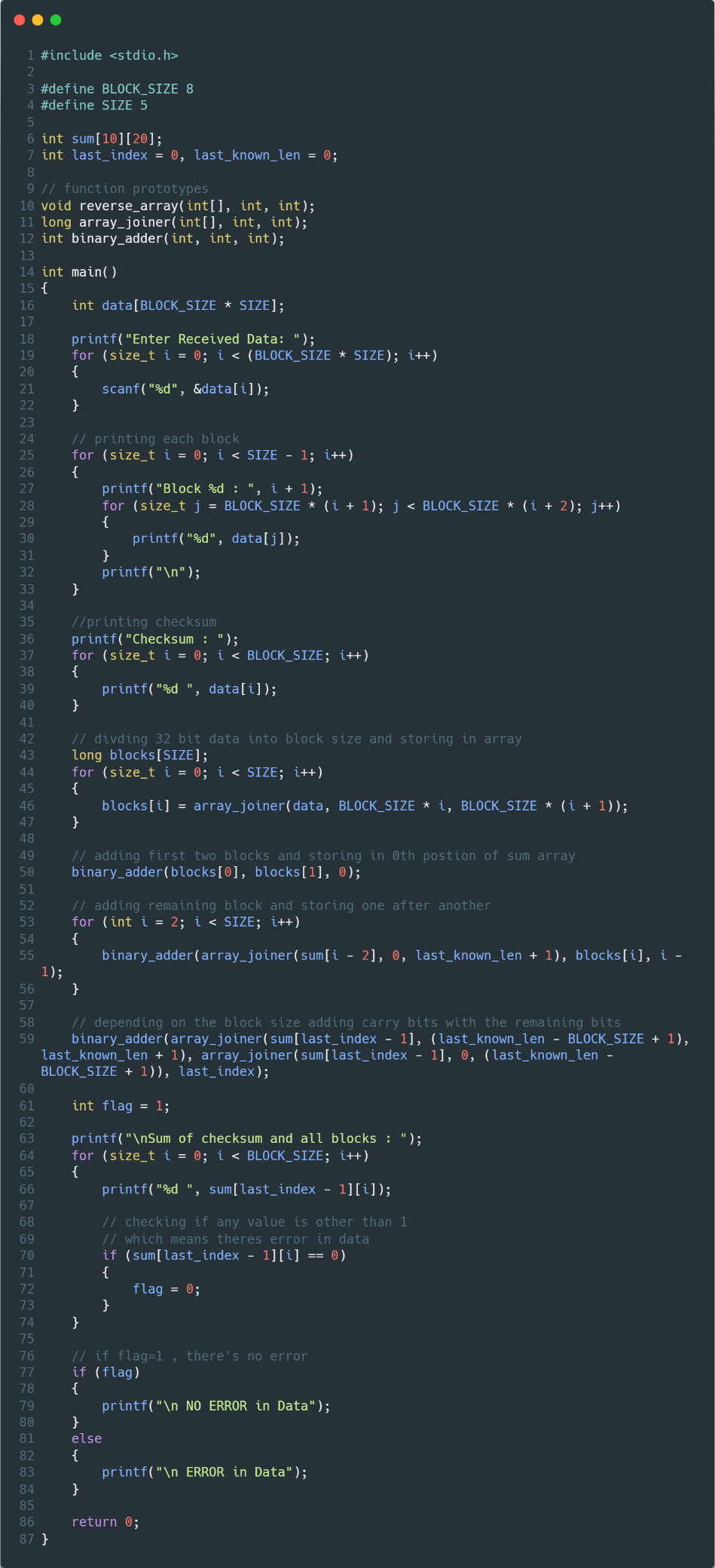


Figure driver program



Figure 16 function to reverse an array, join an array and add two binary numbers

Output

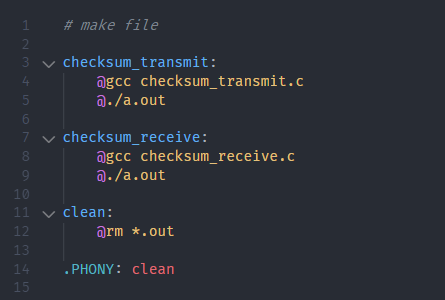


Figure make file

Let’s say we received data:

Where,

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Block 1 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 |
| Block 2 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| Block 3 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 |
| Block 4 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 |
| checksum | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 1 |

Adding all the blocks and checksum we get

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Block 1 |  |  | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 |
| Block 2 |  |  | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| Block 3 |  |  | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 |
| Block 4 |  |  | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 |
| checksum |  | **+** | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 1 |
|  |  |  |  |  |  |  |  |  |  |  |
| Sum |  |  | **1** | **1** | **1** | **1** | **1** | **1** | **1** | **1** |

Hence, the data is error free.

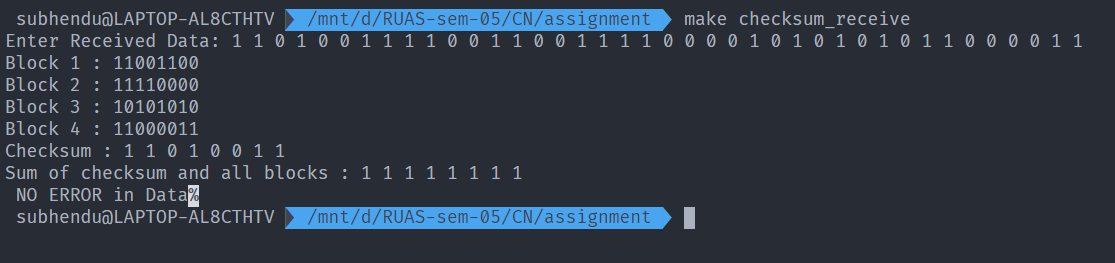


Figure execution of checksum\_receive with correct received data

Running the program with last bit changed,

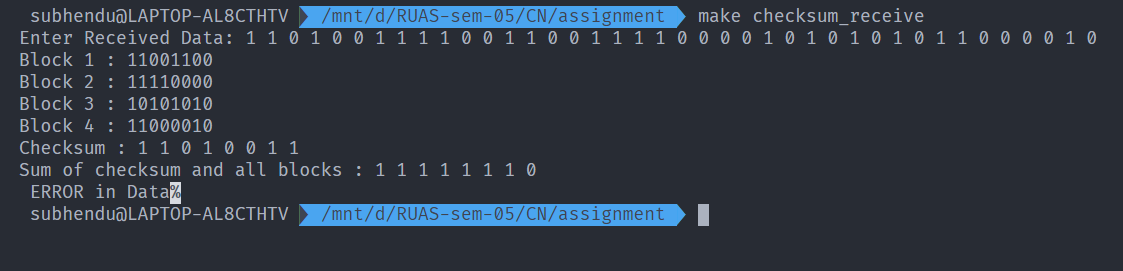


Figure execution of checksum\_receive with incorrect received data

**Advantage of checksum**

The checksum detects all the errors involving an odd number of bits as well as the error involving an even number of bits.

**Disadvantage of checksum**

The main problem is that the error goes undetected if one or more bits of a subunit is damaged and the corresponding bit or bits of a subunit are damaged and the corresponding bit or bits of opposite value in second subunit are also damaged. This is because the sum of those columns remains unchanged.

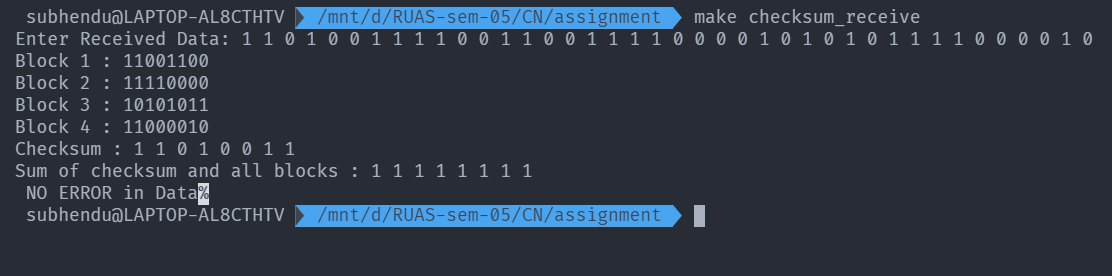
Example:

Let’s say we have data:

Where,

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Block 1 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 |
| Block 2 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| Block 3 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 |
| Block 4 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 |
| checksum | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 1 |

But the data received is



Although data is corrupted, the error is undetected.

**Bibliography**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. <https://www.javatpoint.com/stop-and-wait-protocol>
2. <https://www.geeksforgeeks.org/stop-and-wait-arq/>
3. https://www.tutorialspoint.com/error-detecting-codes-checksums