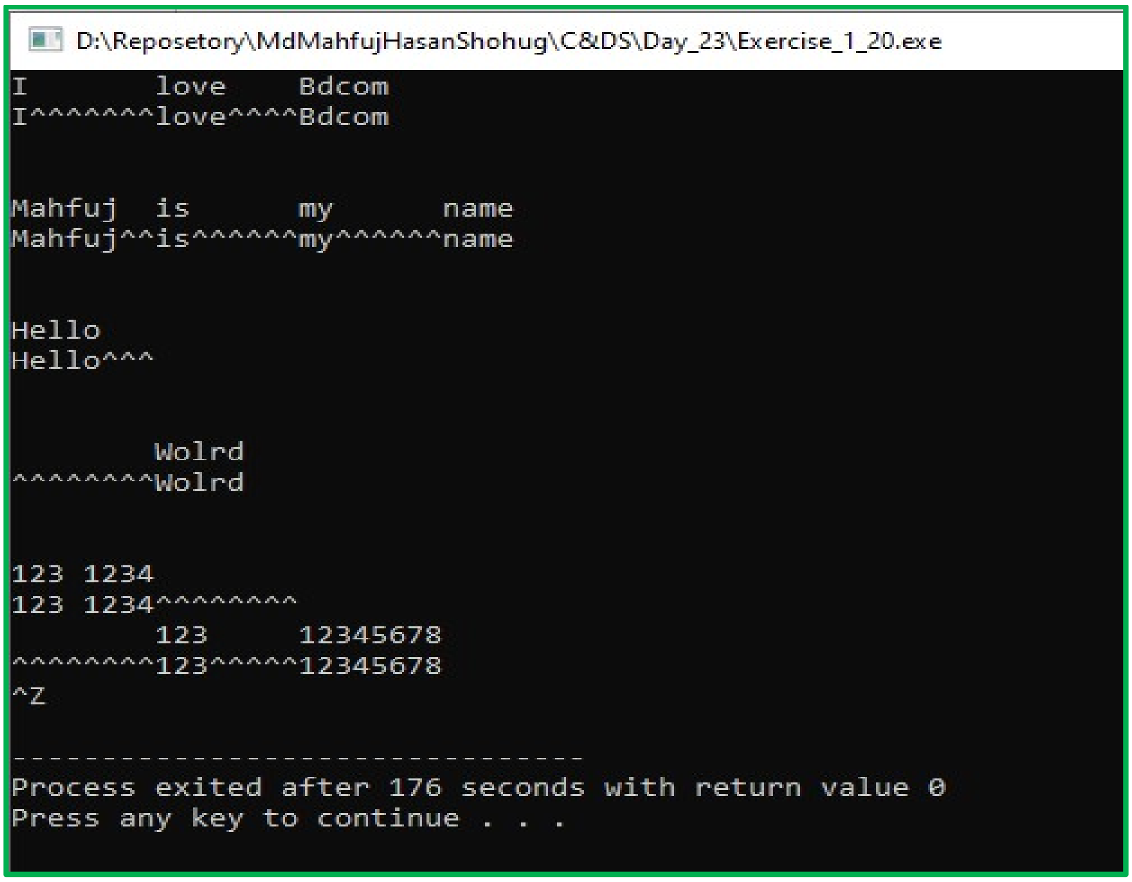
Day 23 Documentation

Md. Mahfuj Hasan Shohug

BDCOM0019

1. **Exercise 1-20:** Write a program detab that replaces tabs in the input with the proper number of blanks to space to the next tab stop. Assume a fixed set of tab stops, say every n columns. Should n be a variable or a symbolic parameter?

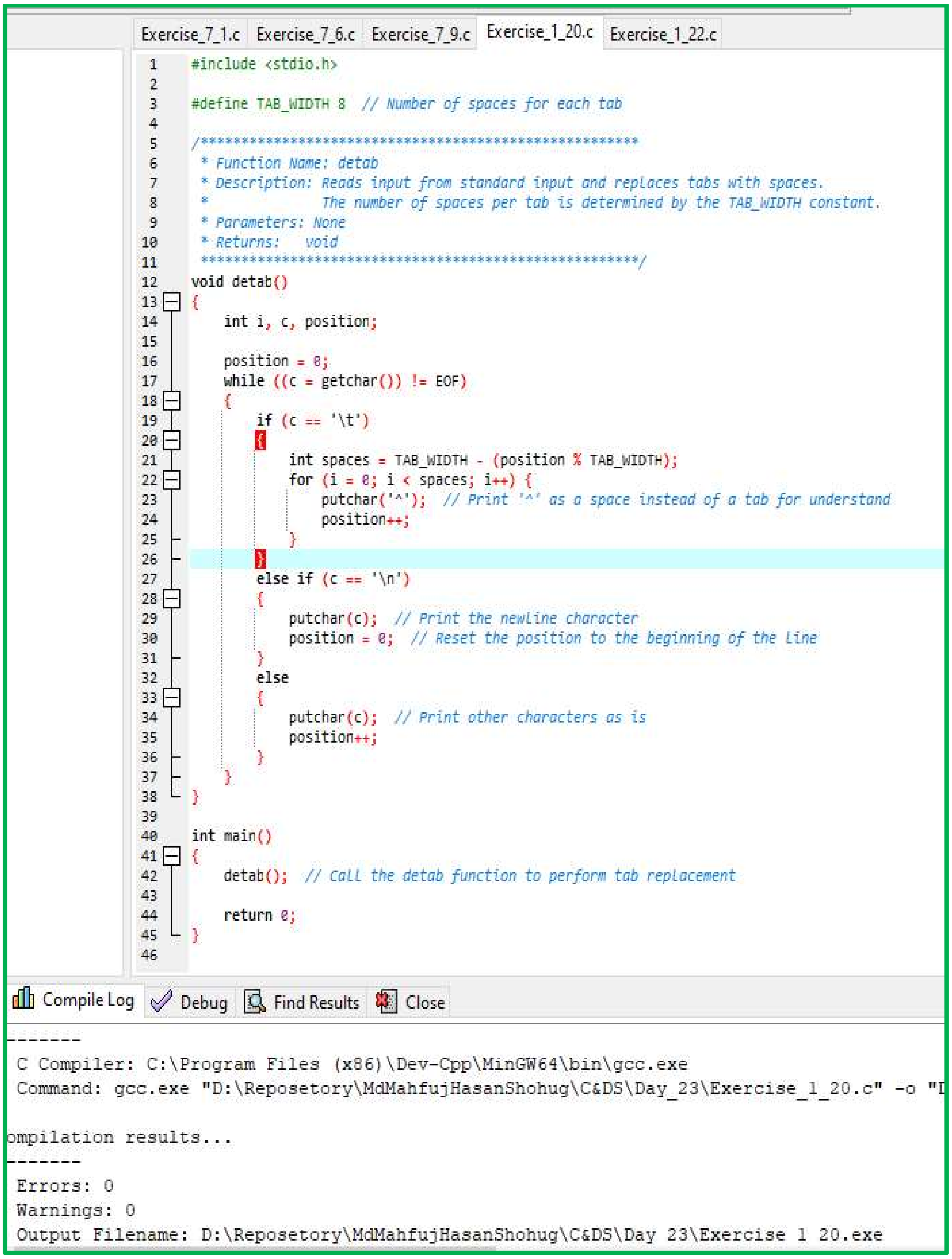
   Answer: The program at hand reads input from the standard input and replacements tab characters with spaces. The TAB_WIDTH constant, which is set to 8 in this code, controls how many spaces there are on each tab. The detab() function scans the input for tab characters while reading characters. Based on the current position, it determines the amount of spaces required to get to the next tab stop when a tab is encountered. It then prints the calculated number of spaces **by using the putchar() function to output '^'** instead of a tab character.
   If a newline character appears, it is displayed exactly as it is, and the line's position is then reset to the beginning. The position is increased appropriately and additional characters are printed exactly as they are.
   Here is some test case on this program:



D:\Reposetory\MdMahfujHasanShohug\C&DS\Day_23\Exercise_1_20.exe

```
I       love    Bdcom
I^^^^^^^love^^^^Bdcom


Mahfuj  is      my      name
Mahfuj^^is^^^^^^my^^^^^^name


Hello
Hello^^^


        Wolrd
^^^^^^^^Wolrd


123 1234
123 1234^^^^^^^^
        123     12345678
^^^^^^^^123^^^^^12345678
^Z

--------------------------------
Process exited after 176 seconds with return value 0
Press any key to continue . . .
```

Source Code:

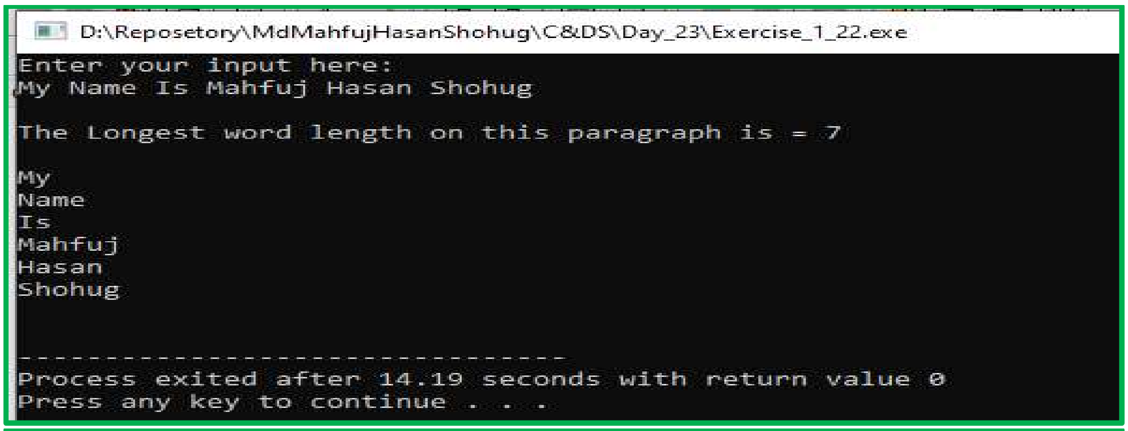


```c
#include <stdio.h>

#define TAB_WIDTH 8  // Number of spaces for each tab

/******************************************************
 * Function Name: detab
 * Description: Reads input from standard input and replaces tabs with spaces.
 *              The number of spaces per tab is determined by the TAB_WIDTH constant.
 * Parameters: None
 * Returns:   void
 ******************************************************/
void detab()
{
    int i, c, position;

    position = 0;
    while ((c = getchar()) != EOF)
    {
        if (c == '\t')
        {
            int spaces = TAB_WIDTH - (position % TAB_WIDTH);
            for (i = 0; i < spaces; i++) {
                putchar('^');  // Print '^' as a space instead of a tab for understand
                position++;
            }
        }
        else if (c == '\n')
        {
            putchar(c);  // Print the newline character
            position = 0;  // Reset the position to the beginning of the line
        }
        else
        {
            putchar(c);  // Print other characters as is
            position++;
        }
    }
}

int main()
{
    detab();  // Call the detab function to perform tab replacement

    return 0;
}
```

```
-------
 C Compiler: C:\Program Files (x86)\Dev-Cpp\MinGW64\bin\gcc.exe
 Command: gcc.exe "D:\Reposetory\MdMahfujHasanShohug\C&DS\Day_23\Exercise_1_20.c" -o "I

ompilation results...
-------
 Errors: 0
 Warnings: 0
 Output Filename: D:\Reposetory\MdMahfujHasanShohug\C&DS\Day 23\Exercise 1 20.exe
```

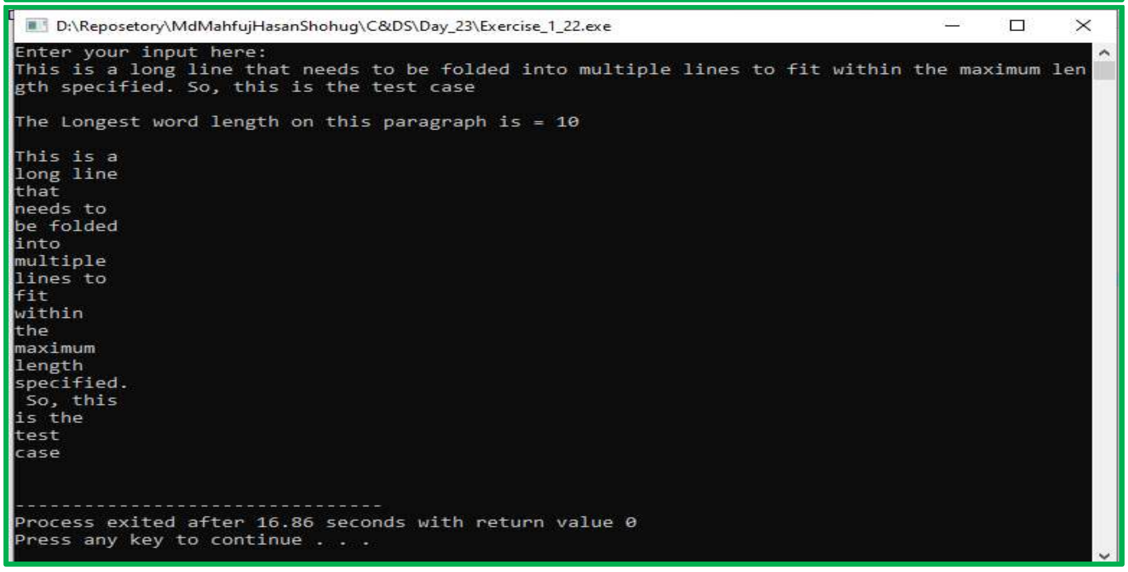
2. **Exercise 1-22.** Write a program to ``fold'' long input lines into two or more shorter lines after the last non-blank character that occurs before the n-th column of input. Make sure your program does something intelligent with very long lines, and if there are no blanks or tabs before the specified column.
Answer: Find the longest word in a line by using the function find_longest_word_length. The function loops through each letter in the line and counts each string of characters that aren't a space or tab as a word. The longest word length that has been used so far is recorded and returned.
fold_lines: This function divides a line into numerous lines if it is longer than a predetermined maximum length. It chooses a suitable breaking point (space or tab) within the limit length by starting at the beginning of the line. The folded lines are then printed.
In conclusion, the program analyzes the length of the longest word in a given line and, if it is longer than a predetermined length, folds the line into numerous lines.

Test case on this code:



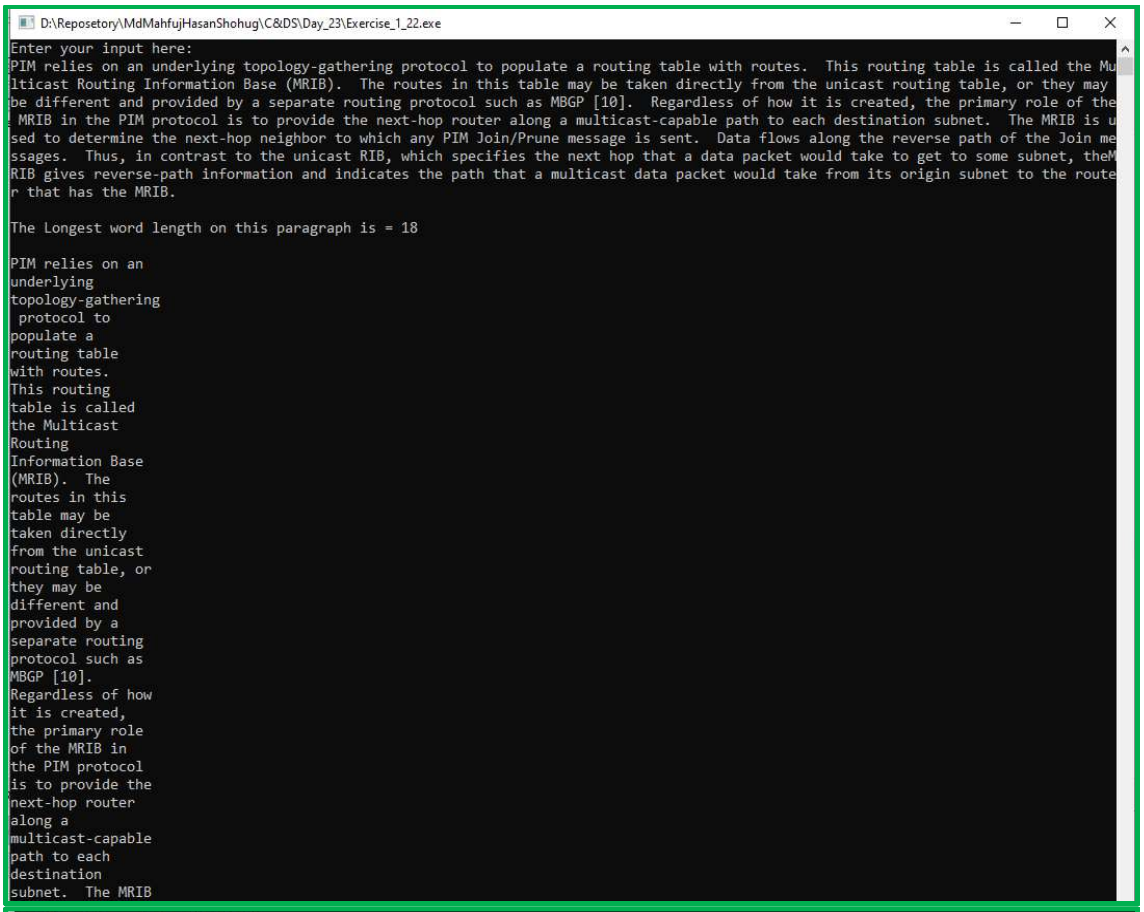
D:\Reposetory\MdMahfujHasanShohug\C&DS\Day_23\Exercise_1_22.exe

```
Enter your input here:
My Name Is Mahfuj Hasan Shohug

The Longest word length on this paragraph is = 7

My
Name
Is
Mahfuj
Hasan
Shohug


--------------------------------
Process exited after 14.19 seconds with return value 0
Press any key to continue . . .
```



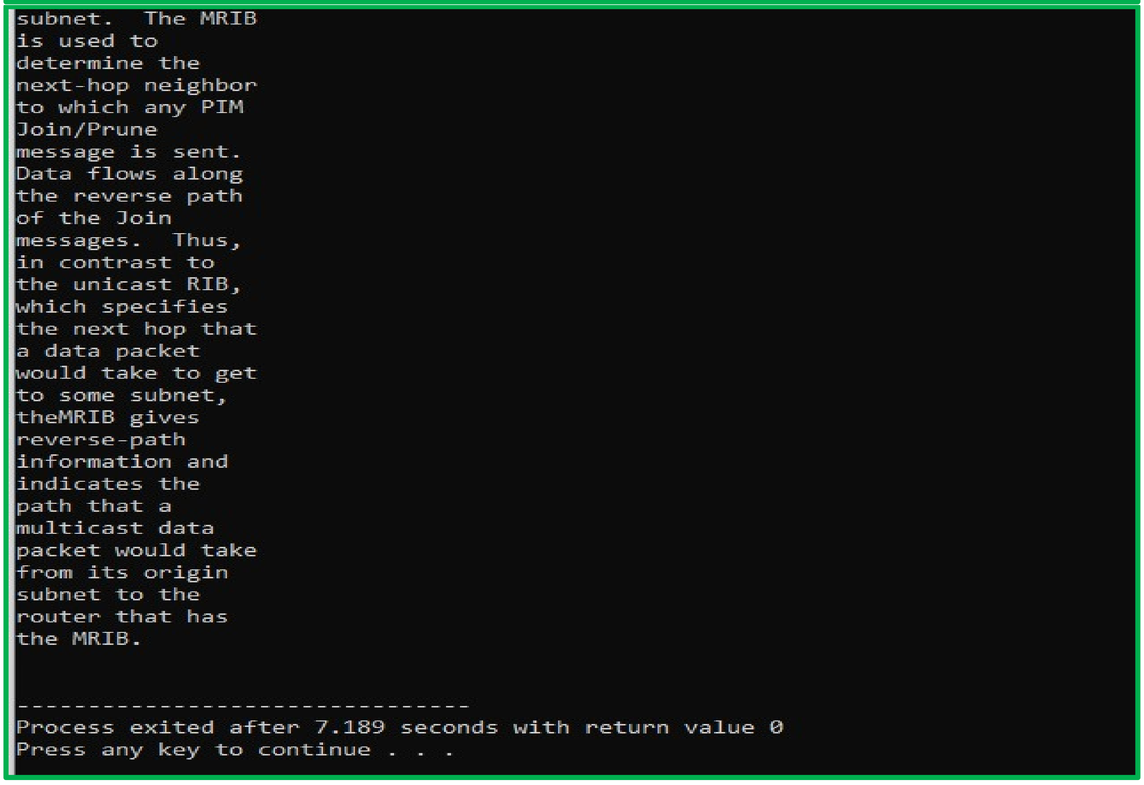
D:\Reposetory\MdMahfujHasanShohug\C&DS\Day_23\Exercise_1_22.exe

```
Enter your input here:
This is a long line that needs to be folded into multiple lines to fit within the maximum len
gth specified. So, this is the test case

The Longest word length on this paragraph is = 10

This is a
long line
that
needs to
be folded
into
multiple
lines to
fit
within
the
maximum
length
specified.
 So, this
is the
test
case


--------------------------------
Process exited after 16.86 seconds with return value 0
Press any key to continue . . .
```



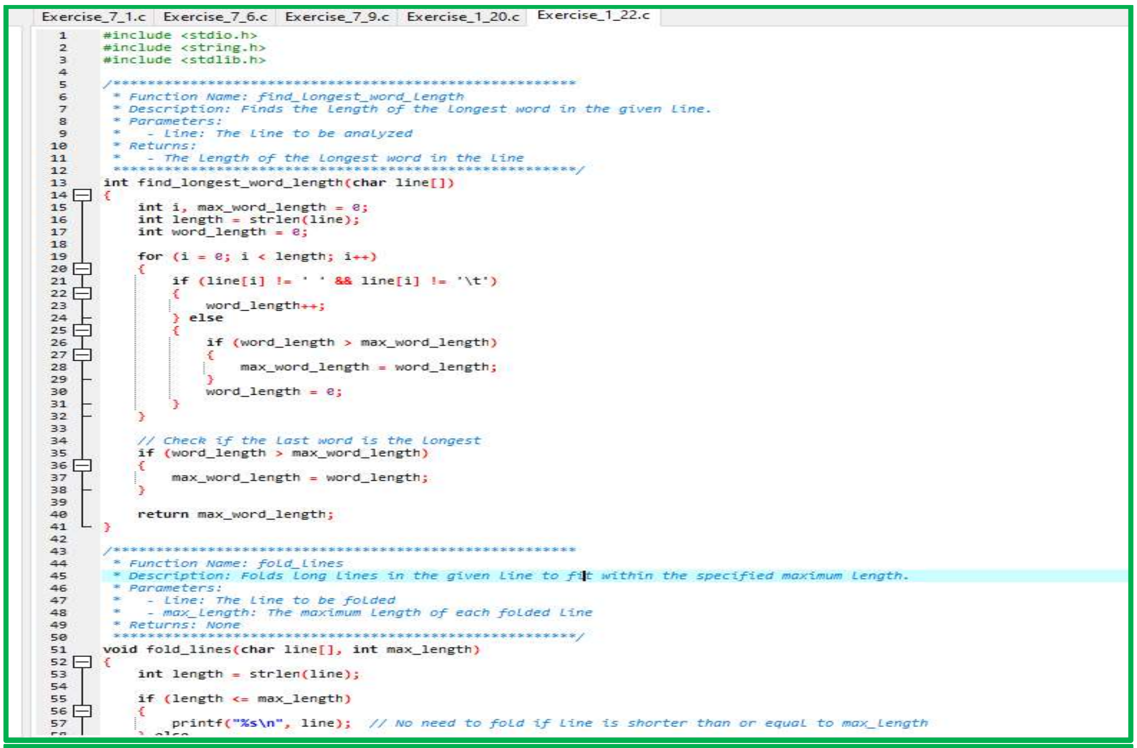
D:\Reposetory\MdMahfujHasanShohug\C&DS\Day_23\Exercise_1_22.exe

```
Enter your input here:
PIM relies on an underlying topology-gathering protocol to populate a routing table with routes.  This routing table is called the Mu
lticast Routing Information Base (MRIB).  The routes in this table may be taken directly from the unicast routing table, or they may
be different and provided by a separate routing protocol such as MBGP [10].  Regardless of how it is created, the primary role of the
 MRIB in the PIM protocol is to provide the next-hop router along a multicast-capable path to each destination subnet.  The MRIB is u
sed to determine the next-hop neighbor to which any PIM Join/Prune message is sent.  Data flows along the reverse path of the Join me
ssages.  Thus, in contrast to the unicast RIB, which specifies the next hop that a data packet would take to get to some subnet, theM
RIB gives reverse-path information and indicates the path that a multicast data packet would take from its origin subnet to the route
r that has the MRIB.

The Longest word length on this paragraph is = 18

PIM relies on an
underlying
topology-gathering
 protocol to
populate a
routing table
with routes.
This routing
table is called
the Multicast
Routing
Information Base
(MRIB).  The
routes in this
table may be
taken directly
from the unicast
routing table, or
they may be
different and
provided by a
separate routing
protocol such as
MBGP [10].
Regardless of how
it is created,
the primary role
of the MRIB in
the PIM protocol
is to provide the
next-hop router
along a
multicast-capable
path to each
destination
subnet.  The MRIB
```

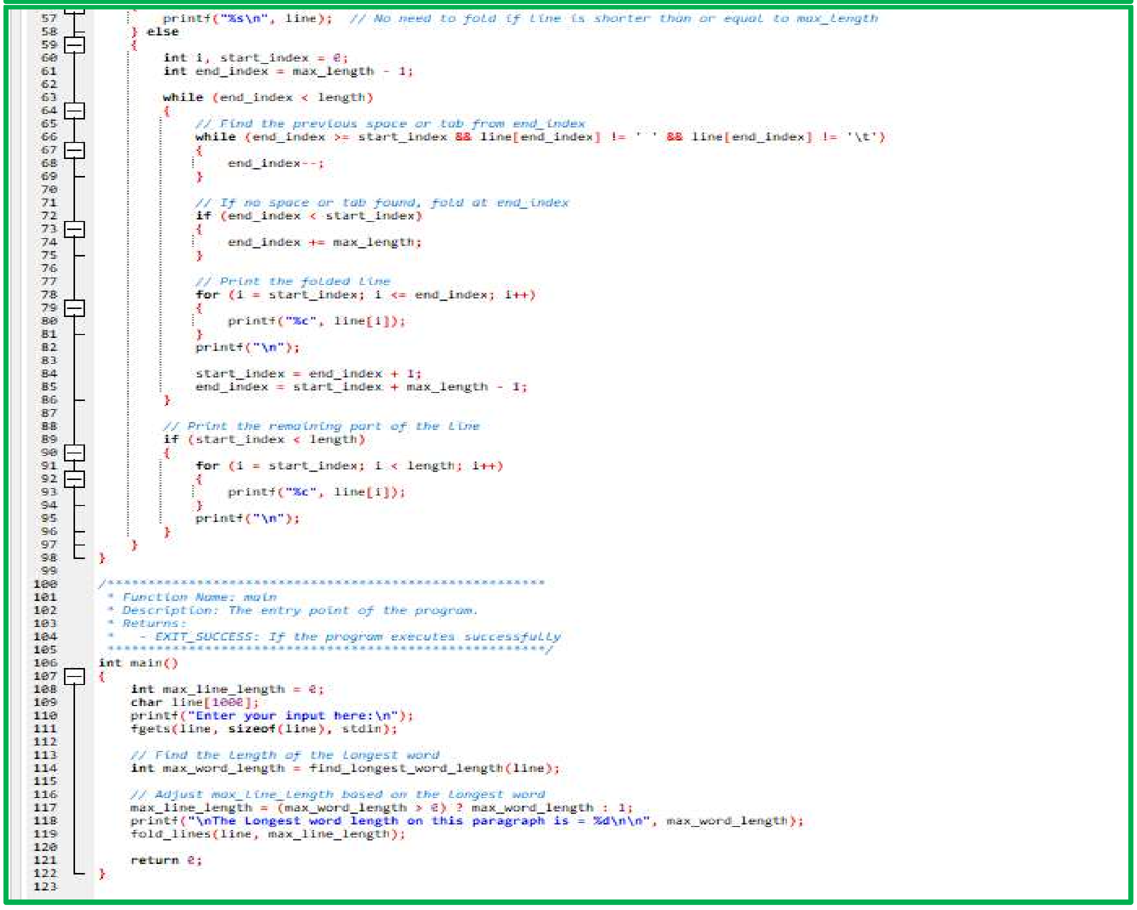


```
subnet.  The MRIB
is used to
determine the
next-hop neighbor
to which any PIM
Join/Prune
message is sent.
Data flows along
the reverse path
of the Join
messages.  Thus,
in contrast to
the unicast RIB,
which specifies
the next hop that
a data packet
would take to get
to some subnet,
theMRIB gives
reverse-path
information and
indicates the
path that a
multicast data
packet would take
from its origin
subnet to the
router that has
the MRIB.


--------------------------------
Process exited after 7.189 seconds with return value 0
Press any key to continue . . .
```

Source Code:



```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

/**********************************************************
 * Function Name: find_longest_word_length
 * Description: Finds the length of the longest word in the given line.
 * Parameters:
 *   - line: The line to be analyzed
 * Returns:
 *     - The length of the longest word in the line
 **********************************************************/
int find_longest_word_length(char line[])
{
    int i, max_word_length = 0;
    int length = strlen(line);
    int word_length = 0;

    for (i = 0; i < length; i++)
    {
        if (line[i] != ' ' && line[i] != '\t')
        {
            word_length++;
        } else
        {
            if (word_length > max_word_length)
            {
                max_word_length = word_length;
            }
            word_length = 0;
        }
    }

    // Check if the last word is the longest
    if (word_length > max_word_length)
    {
        max_word_length = word_length;
    }

    return max_word_length;
}

/**********************************************************
 * Function Name: fold_lines
 * Description: Folds long lines in the given line to fit within the specified maximum length.
 * Parameters:
 *   - line: The line to be folded
 *   - max_length: The maximum length of each folded line
 * Returns: None
 **********************************************************/
void fold_lines(char line[], int max_length)
{
    int length = strlen(line);

    if (length <= max_length)
    {
        printf("%s\n", line);  // No need to fold if line is shorter than or equal to max_length
```



```c
        printf("%s\n", line);  // No need to fold if line is shorter than or equal to max_length
    } else
    {
        int i, start_index = 0;
        int end_index = max_length - 1;

        while (end_index < length)
        {
            // Find the previous space or tab from end_index
            while (end_index >= start_index && line[end_index] != ' ' && line[end_index] != '\t')
            {
                end_index--;
            }

            // If no space or tab found, fold at end_index
            if (end_index < start_index)
            {
                end_index += max_length;
            }

            // Print the folded line
            for (i = start_index; i <= end_index; i++)
            {
                printf("%c", line[i]);
            }
            printf("\n");

            start_index = end_index + 1;
            end_index = start_index + max_length - 1;
        }

        // Print the remaining part of the line
        if (start_index < length)
        {
            for (i = start_index; i < length; i++)
            {
                printf("%c", line[i]);
            }
            printf("\n");
        }
    }
}

/**********************************************************
 * Function Name: main
 * Description: The entry point of the program.
 * Returns:
 *     - EXIT_SUCCESS: If the program executes successfully
 **********************************************************/
int main()
{
    int max_line_length = 0;
    char line[1000];
    printf("Enter your input here:\n");
    fgets(line, sizeof(line), stdin);

    // Find the length of the longest word
    int max_word_length = find_longest_word_length(line);

    // Adjust max_line_length based on the longest word
    max_line_length = (max_word_length > 0) ? max_word_length : 1;
    printf("\nThe Longest word length on this paragraph is = %d\n\n", max_word_length);
    fold_lines(line, max_line_length);

    return 0;
}
```