

Documentation of Day 6

Exercise 2-1:

Question: Write a program to determine the ranges of char, short, int, and long variables, both signed and unsigned, by printing appropriate values from standard headers and by direct computation. Harder if you compute them: determine the ranges of the various floating-point types.

Source Code for direct computation:

```
#include <stdio.h>
#include <math.h>
#include <limits.h>
int main(){
    printf("----- INTEGER -----\n");
    printf("Integer unsigned maximum value: %u\n", (unsigned)(pow(2, 32)-1));
    printf("Integer unsigned minimum value: %u\n", (unsigned)0);
    printf("Integer signed maximum value: %d\n", (signed)(pow(2, 31)-1));
    printf("Integer signed minimum value: %d\n", (signed)(-pow(2, 31)));
    printf("----- SHORT INTEGER -----\n");
    printf("Short Integer unsigned maximum value: %u\n", (unsigned)(pow(2, 16)-1));
    printf("Short Integer unsigned minimum value: %d\n", (unsigned)0);
    printf("Short Integer signed maximum value: %d\n", (signed)(pow(2, 16)-1));
    printf("Short Integer signed minimum value: %d\n", (signed)(-pow(2, 16)));
    printf("----- LONG INTEGER -----\n");

    printf("Long Integer unsigned maximum value: %lu\n", (unsigned long)(pow(2, 64)-1));
    printf("Long Integer unsigned minimum value: %lu\n", (unsigned long)0);
    printf("Long Integer signed maximum value: %ld\n", (signed long)(pow(2, 64)-1));
    printf("Long Integer signed minimum value: %ld\n", (signed long)(-pow(2, 64)));

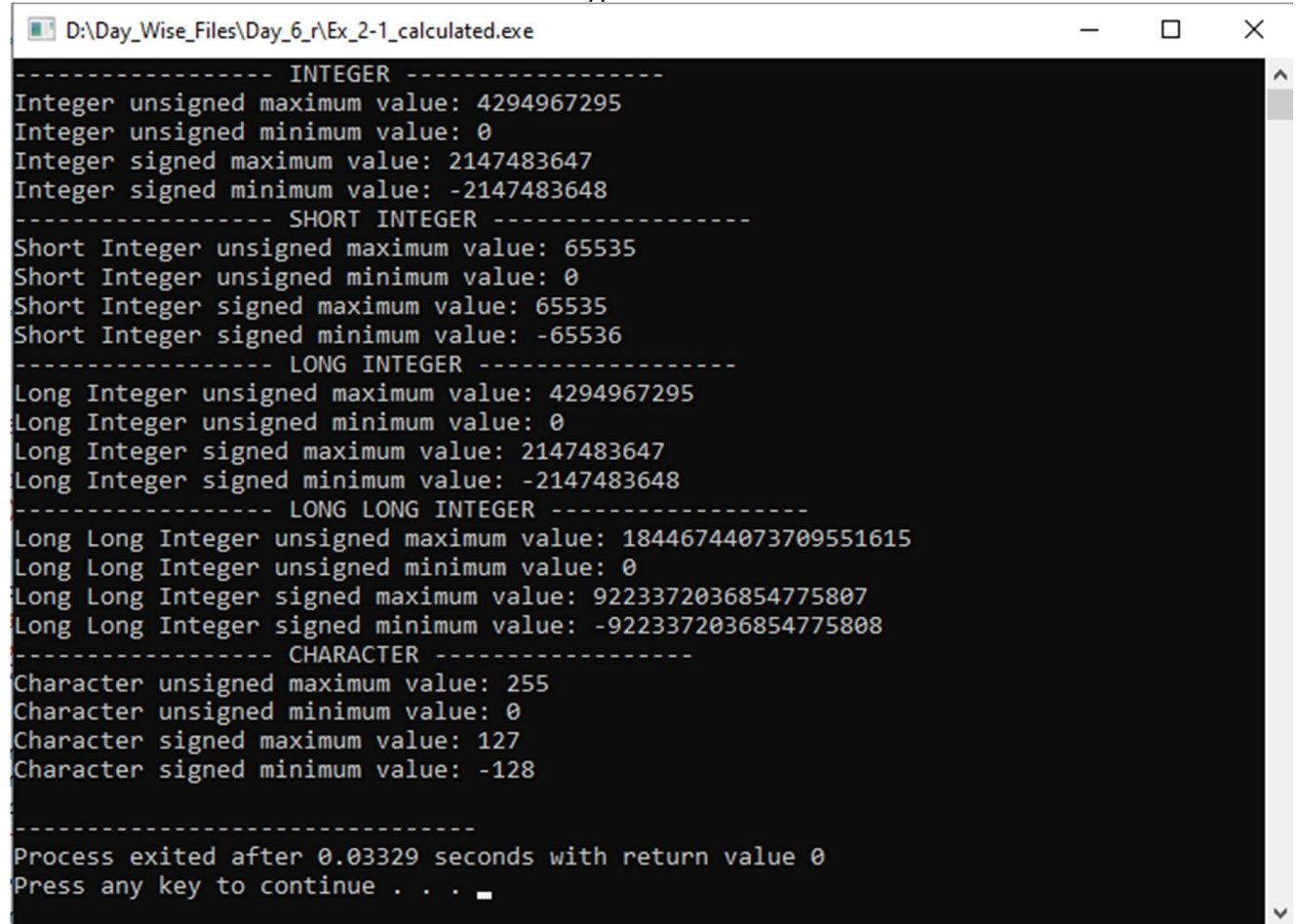
    printf("----- LONG LONG INTEGER -----\n");
    printf("Long Long Integer unsigned maximum value: %llu\n", (unsigned long long)(pow(2, 64)-1));
    printf("Long Long Integer unsigned minimum value: %llu\n", (unsigned long long)0);
    printf("Long Long Integer signed maximum value: %lld\n", (signed long long)(pow(2, 64)-1));
    printf("Long Long Integer signed minimum value: %lld\n", (signed long long)(-pow(2, 64)-1));
    printf("----- CHARACTER -----\n");
    printf("Character unsigned maximum value: %d\n", (unsigned)(pow(2, 8)-1));
    printf("Character unsigned minimum value: %d\n", (unsigned)0);
    printf("Character signed maximum value: %d\n", (signed)(pow(2, 7)-1));
    printf("Character signed minimum value: %d\n", (signed)(-pow(2, 7)));
    return 0;
}
```

Explanation:

1. The above code calculates and prints the minimum and maximum value ranges for different integer types (int, short int, long int, long long int) and the character type.
2. Here is the step by step explanation of the above source code:
3. In order to include the essential header files, the program begins with #include statements. These include stdio.h for input/output functions, math.h for mathematical functions, and limits.h for constants pertaining to integer types.
4. The program's entry point, the main function, has been defined.
5. The ranges for the int type are then printed by the program
6. Messages describing the kind of integer being evaluated (such as "Integer unsigned maximum value," "Integer unsigned minimum value," etc.) are printed with the calculated values using the printf function.
7. The maximum and minimum values are computed using the pow function from the math.h library. For instance, by raising 2 to the power of 32 and taking away 1, (unsigned)(pow(2, 32)-1) determines the highest unsigned value for a 32-bit integer.
8. The calculated values are formatted and printed using printf and the format specifiers %u, %d, %lu, %ld, %llu, etc.
9. The short int, long int, long long int, and char type ranges are evaluated and printed using a similar method in the code.
10. The main function and the program are terminated by the return 0 command.

Output:

By running the program, we can see the output printed on the console, which shows the maximum and minimum values for each data type.



```
D:\Day_Wise_Files\Day_6_r\Ex_2-1_calculated.exe

----- INTEGER -----
Integer unsigned maximum value: 4294967295
Integer unsigned minimum value: 0
Integer signed maximum value: 2147483647
Integer signed minimum value: -2147483648
----- SHORT INTEGER -----
Short Integer unsigned maximum value: 65535
Short Integer unsigned minimum value: 0
Short Integer signed maximum value: 65535
Short Integer signed minimum value: -65536
----- LONG INTEGER -----
Long Integer unsigned maximum value: 4294967295
Long Integer unsigned minimum value: 0
Long Integer signed maximum value: 2147483647
Long Integer signed minimum value: -2147483648
----- LONG LONG INTEGER -----
Long Long Integer unsigned maximum value: 18446744073709551615
Long Long Integer unsigned minimum value: 0
Long Long Integer signed maximum value: 9223372036854775807
Long Long Integer signed minimum value: -9223372036854775808
----- CHARACTER -----
Character unsigned maximum value: 255
Character unsigned minimum value: 0
Character signed maximum value: 127
Character signed minimum value: -128

-----
Process exited after 0.03329 seconds with return value 0
Press any key to continue . . .
```

Source Code for printing values from standard headers:

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

int main(void)
{
    printf("----- CHARACTER -----\\n");
    printf("bits: %d\\n", CHAR_BIT);
    printf("unsigned char max: %d\\n", UCHAR_MAX);
    printf("signed char min: %d\\n", SCHAR_MIN);
    printf("signed char max: %d\\n", SCHAR_MAX);
    printf("\\n");

    printf("----- INTEGER -----\\n");
    printf("unsigned int max: %u\\n", UINT_MAX);
    printf("signed int min: %d\\n", INT_MIN);
    printf("signed int max: %d\\n", INT_MAX);
    printf("\\n");

    printf("----- SHORT INTEGER -----\\n");
    printf("unsigned short int max: %u\\n", USHRT_MAX);
    printf("signed short int min: %d\\n", SHRT_MIN);
    printf("signed short int max: %d\\n", SHRT_MAX);
    printf("\\n");

    printf("----- LONG INTEGER -----\\n");
    printf("unsigned long int max: %lu\\n", ULONG_MAX);
    printf("signed long int min: %ld\\n", LONG_MIN);
    printf("signed long int max: %ld\\n", LONG_MAX);
    printf("\\n");

    printf("----- LONG LONG INTEGER -----\\n");
    printf("unsigned long long int max: %llu\\n", ULLONG_MAX);
    printf("signed long long int min: %lld\\n", LLONG_MIN);
    printf("signed long long int max: %lld\\n", LLONG_MAX);
    printf("\\n");

    return 0;
}
```

The program I have given above prints the size and range limits of various integer types in C, including char, int, short int, long int, and long long int.

Here is the explanation of this program:

1. The program includes the necessary header file `stdio.h` for input/output functions and `limits.h` for constants related to integer types.
2. The main function is defined, which is the entry point of the program.
3. The program then proceeds to print information about the char type.
4. The `printf` function is used to print messages indicating the type being evaluated (e.g., "CHAR", "INT", "SHORT INT", etc.) along with the calculated values.
5. The `%d`, `%u`, `%ld`, `%lu`, `%lld`, etc., format specifiers are used with `printf` to format and print the values of various constants defined in the `limits.h` header.
6. The code follows a similar pattern for printing information about the int, short int, long int, and long long int types.
7. The program uses specific constants provided by the `limits.h` header to determine the size and range of each integer type.

Finally, the program returns 0, indicating successful execution of the program.

Exercise 2-2:

Question: Write a loop equivalent to the for loop above without using `&&` or `||`.

Source Code:

```
#include <stdio.h>
#define LIMIT 50
int main(){
    int i, c;
    char s[LIMIT];

    printf("Enter the string: ");
    for(i = 0; ; ++i){
        if(i > (LIMIT-1))
            break;
        c = getchar();

        if(c == '\n')
            break;

        else if(c==EOF)
```

```

        break;
    else
        s[i] = c;
}

s[i] = '\0';
printf("String is: %s\n", s);
return 0;
}

```

Explanation:

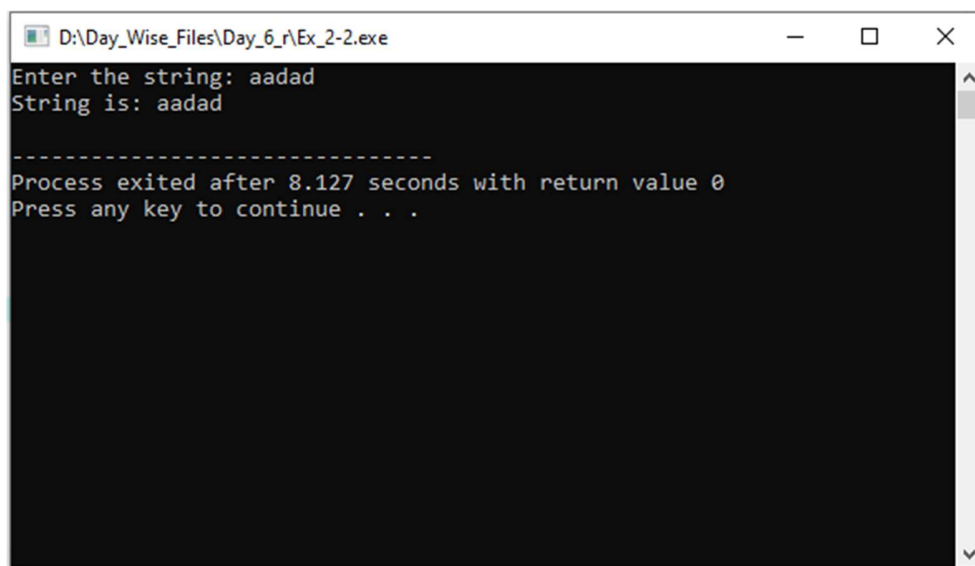
The given code according to this exercise has a for loop which is used in the getline function. It checks for three conditions in its declaration: `i lim - 1`, `(c=getchar())!= EOF`, and `c!= "n"`. As long as all three conditions are met, the loop keeps running.

The main function's for loop in my code is missing the criteria from its declaration. Instead, it has an unconditional loop condition that will always be true (for `(i = 0; ; ++i)`), causing the loop to run endlessly until a break statement is found.

If conditions are used inside the loop to deal with the situations where `i` exceeds the `LIMIT-1` limit, when character `c` is a newline character (`'n'`), and where character `c` is EOF (end-of-file). The break statement is used to end the loop in each of these scenarios.

The modified version of the code that I have written gives the same result as all the loop controlled structures and conditions were handled properly along with the proper break conditions.

Output:



```

D:\Day_Wise_Files\Day_6_r\Ex_2-2.exe
Enter the string: aadad
String is: aadad

-----
Process exited after 8.127 seconds with return value 0
Press any key to continue . . .

```

Exercise 2-3:

Question: Write a function `htoi(s)`, which converts a string of hexadecimal digits (including an optional `0x` or `0X`) into its equivalent integer value. The allowable digits are 0 through 9, a through f, and A through F.

The source code of implementation of the 'htoi' function that converts a string of hexadecimal digits into its equivalent integer value is given below:

Source Code:

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

int htoi(const char *s) {
    int result = 0;
    int i = 0;

    // Skip optional 0x or 0X prefix
    if (s[i] == '0' && (s[i+1] == 'x' || s[i+1] == 'X'))
        i += 2;

    // Iterate through each character of the string
    while (s[i] != '\0') {
        char c = tolower(s[i]);
        int digit;

        // Convert hexadecimal digit to decimal value
        if (isdigit(c))
            digit = c - '0';
        else if (c >= 'a' && c <= 'f')
            digit = c - 'a' + 10;
        else {
            // Invalid character encountered
            printf("Invalid hexadecimal digit '%c'\n", s[i]);
            return 0;
        }

        // Accumulate the result
        result = result * 16 + digit;
        i++;
    }

    return result;
}
```

```

}

int main() {
    char hex[20];

    printf("Enter a hexadecimal number: ");
    scanf("%s", hex);

    int decimal = htoi(hex);
    printf("Decimal equivalent: %d\n", decimal);

    return 0;
}

```

Explanation:

The htoi function takes a string s as input and iterates through each character of the string. It checks for an optional "0x" or "0X" prefix and skips it if present. Then, it converts each hexadecimal digit to its decimal value using the tolower and isdigit functions, and accumulates the result by multiplying it by 16 and adding the digit value. The function returns the final result as an integer.

In the main function, the user is prompted to enter a hexadecimal number, and the htoi function is called to convert it to decimal. The decimal equivalent is then printed.

Here is the step by step procedure of the above code to understand the functionality in detail:

At first the code includes the necessary header files for input/output operations (stdio.h) and character handling functions (ctype.h).

The htoi function takes a string s as input and returns its equivalent decimal value as an integer.

1. It defines and initializes the result variable, which will hold the final decimal number, as well as the i variable, which will be used to loop through the string's characters.
2. It determines whether a string's optional "0x" or "0X" prefix is present. In order to skip the prefix, the i value is incremented by 2 if it is present.
3. The function starts a while loop and continues iterating until it reaches the string's end ('0').
4. The tolower function is used inside the loop to lowercase the current character, s[i], and it is then saved in the c variable. To handle both uppercase and lowercase hexadecimal numbers consistently, this step is carried out.
5. Using the isdigit function, the function determines whether c represents a decimal digit. If this is the case, the ASCII value of '0' is subtracted from c to get the digit's decimal value. For instance, the decimal value of the digit is 9 if the value of c is "9."
6. If c is not a decimal digit, the function uses the condition c >= 'a' && c <= 'f' to determine if it represents a lowercase hexadecimal digit ('a' to 'f'). If this is the case, the decimal value of the digit is obtained by subtracting the ASCII value of 'a' and adding 10 to it. For instance, the decimal value of the digit c if it is "d" is 13.
7. The function produces a message describing the invalid character and returns -1 if the character s[i] is an invalid hexadecimal digit.
8. The function then adds the decimal value of the current digit to the result of multiplying the result value by 16 and accumulating the decimal value.
9. In order to advance to the following character in the string, it finally increments i.
10. The function then returns the total result, which is the hexadecimal string's decimal representation, when the loop has completed.

Output:

i)

```
Enter a hexadecimal number: A
Decimal equivalent: 10
-----
Process exited after 46.76 seconds with return value 0
Press any key to continue . . .
```

ii)

```
D:\Day_Wise_Files\Day_6_r\Ex_2-3.exe
Enter a hexadecimal number: F
Decimal equivalent: 15

-----
Process exited after 3.39 seconds with return value 0
Press any key to continue . . .
```

iii)

```
D:\Day_Wise_Files\Day_6_r\Ex_2-3.exe
Enter a hexadecimal number: H
Invalid hexadecimal digit 'H'
Decimal equivalent: -1

-----
Process exited after 2.893 seconds with return value 0
Press any key to continue . . .
```

Exercise 2-4:

Question: Write an alternative version of `squeeze(s1,s2)` that deletes each character in `s1` that matches any character in the string `s2`.

Source Code:

```
#include <stdio.h>
void squeezeString(char s1[], const char s2[]) {
```

```

int i, j, k;
int shouldDelete;

for (i = j = 0; s1[i] != '\0'; i++) {
    shouldDelete = 0;

    for (k = 0; s2[k] != '\0'; k++) {
        if (s1[i] == s2[k]) {
            shouldDelete = 1;
            break;
        }
    }

    if (!shouldDelete)
        s1[j++] = s1[i];
}

s1[j] = '\0';
}

int main() {
    char s1[100];
    char s2[100];

    printf("Enter the first string (s1): ");
    scanf("%s", s1);

    printf("Enter the second string (s2): ");
    scanf("%s", s2);

    squeezeString(s1, s2);

    printf("Modified string (s1): %s\n", s1);

    return 0;
}

```

Explanation:

The squeezeString method in this variant has two input parameters, s1 and s2. It tries to see if any character in s2 matches each character in s1 as it iterates over the string. If a match is discovered, the character in s1 should be removed; otherwise, the character is retained.

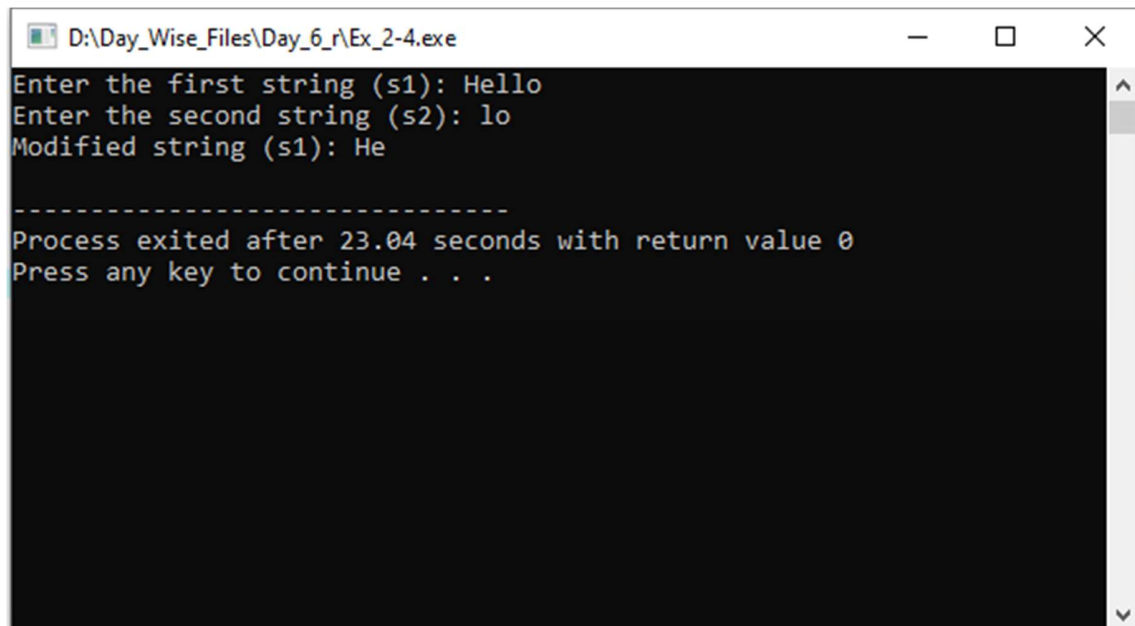
Here's a breakdown of how the function works:

1. Three variables are used by the function: i for iterating through s1, j for keeping track of the changed s1 index, and k for iterating through s2.
2. When s1[i] != "0," it enters a for loop that iterates until s1 is finished (s1[i]).
3. The shouldDelete variable is first set to 0 inside the outer loop to denote that the current character in s1 should not be erased by default.
4. s2's characters are iterated through in a nested for loop.
5. It compares s1[i] and s2[k] inside the nested loop to see if the current character in s1 matches any character in s2.
6. The shouldDelete variable is set to 1, indicating that the current character in s1 should be destroyed, if a match is discovered.
7. It then validates the value of shouldDelete after the nested loop.
8. If shouldDelete is 0, then the current character in s1 should be preserved because no match was detected. Then index j is increased and it is copied to the modified s1 at that position.
9. The operation is repeated for the following character in s1 as the outer loop continues to run.
10. The revised s1 is then terminated by appending the null character "0" at the end.

In the main function, two character arrays s1 and s2 are declared to hold the input strings. The user is prompted to enter the first string (s1) using printf and scanf. The user is then prompted to enter the second string (s2) using printf and scanf. The squeezeString function is called with s1 and s2 as arguments to modify s1 based on the matching characters in s2. Finally, the modified s1 string is printed using printf.

By running the program and entering the strings Hello for s1 and lo for s2, the output would be:

Output:



```
D:\Day_Wise_Files\Day_6_r\Ex_2-4.exe
Enter the first string (s1): Hello
Enter the second string (s2): lo
Modified string (s1): He
-----
Process exited after 23.04 seconds with return value 0
Press any key to continue . . .
```

The character 'l' and 'o' from s1 are removed because they match the characters in s2. Therefore, the resulting modified string is "He".

Exercise 3-5:

Question: Write the function `any(s1,s2)`, which returns the first location in a string `s1` where any character from the string `s2` occurs, or -1 if `s1` contains no characters from `s2`. (The standard library function `strpbrk` does the same job but returns a pointer to the location.)

Source Code:

```
#include <stdio.h>

int any(const char s1[], const char s2[]) {
    int i, j;

    for (i = 0; s1[i] != '\0'; i++) {
        for (j = 0; s2[j] != '\0'; j++) {
            if (s1[i] == s2[j])
                return i;
        }
    }

    return -1;
}

int main() {
    char s1[100];
    char s2[100];

    printf("Enter the first string (s1): ");
    scanf("%s", s1);

    printf("Enter the second string (s2): ");
    scanf("%s", s2);

    int result = any(s1, s2);

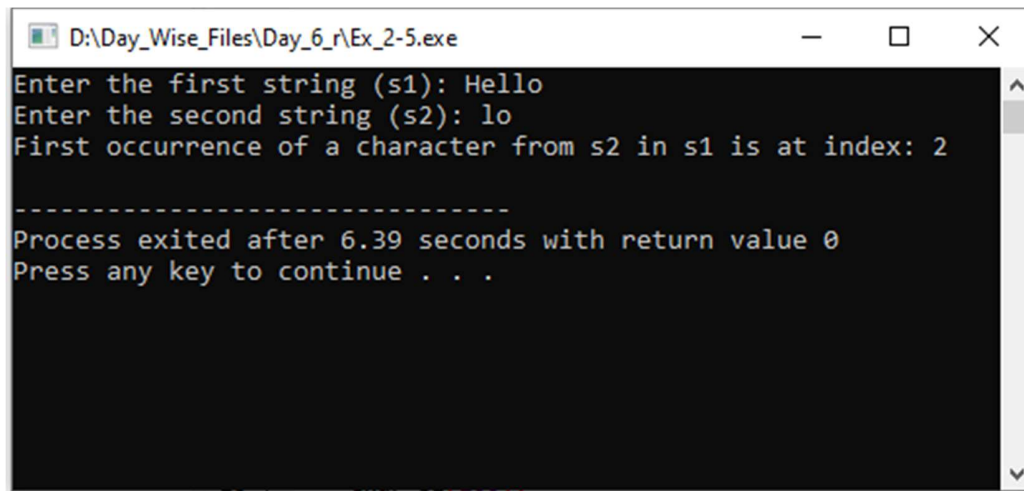
    if (result == -1) {
        printf("No characters from s2 found in s1\n");
    } else {
        printf("First occurrence of a character from s2 in s1 is at index: %d\n", result);
    }

    return 0;
}
```

Explanation:

1. The function iterates across s1 and s2 using two variables, i and j.
2. It then enters an outer for loop that repeats until it reaches the end of s1 (s1[i]!='0').
3. It then starts an inner for loop within the outer loop, iterating there until the end of s2 is reached (s2[j]!='0').
4. The current character in s1 (s1[i]) is compared to any character in s2 (s2[j]) inside the inner loop to see if they match.
5. In the event that a match is made, it instantly returns the index i, which is the first instance of a matching character in the string s1.
6. The inner loop is continued for the following character in s2 if there is no match.
7. When the inner loop is complete, the outer loop is continued for the following character in s1.
8. With this any function, we can find the first occurrence of any character from s2 in s1 and retrieve its location.
9. Two character arrays, s1 and s2, are declared in the main function to hold the input strings.
10. Using printf and scanf, the user is asked to enter the first string (s1).
11. The second string (s2) is then requested from the user using printf and scanf.
12. With s1 and s2 as parameters, the any function is invoked to determine the first instance of any character from s2 in s1.
13. Any function's return value is kept in the result variable.
14. The outcome is examined to see if a match was discovered.
15. A result of -1 indicates that no match was discovered. In this instance, a message stating that s1 had no characters from s2 is printed.

By running the program and entering the strings Hello for s1 and lo for s2, the output would be:

A screenshot of a Windows command prompt window. The title bar shows the file path "D:\Day_Wise_Files\Day_6_r\Ex_2-5.exe" and standard window controls. The command prompt has a black background with white text. It shows the user entering "Hello" for the first string and "lo" for the second string. The program outputs that the first occurrence of a character from s2 in s1 is at index 2. It then displays a separator line of dashes, followed by the message "Process exited after 6.39 seconds with return value 0" and "Press any key to continue . . .".

```
D:\Day_Wise_Files\Day_6_r\Ex_2-5.exe
Enter the first string (s1): Hello
Enter the second string (s2): lo
First occurrence of a character from s2 in s1 is at index: 2

-----
Process exited after 6.39 seconds with return value 0
Press any key to continue . . .
```

The character 'l' from s2 is found at index 2 in s1, which is the first occurrence of a character from s2 in s1. Therefore, the program correctly identifies the location of the match.