2022

Assignment 1 – task 17

SYSTEM PROGRAMMING AND ASSEMBLY

MARTIN ŠVÁB

Assignment:

Write program which allows the user to execute function on input file using the arguments provided when calling said program. When using command option '-h', program must display information about itself and its use. In program use macros with parameters appropriately as well as OS calls for setting the cursor, printing string, clearing screen, file handling etc. Macro definitions must be defined in separate file. Program must be able to handle files of size of at least 64 kB. When reading files use buffer of appropriate size and always read fill the entire buffer until the last read at the end of file. Handle error conditions. Program or every source file must contain necessary technical documentation.

Task 17:

Load 2 strings and print file in such a way, where each occurrence of first string is replaced by the second string. If either of the 2 strings is missing in program arguments, ask for the missing strings in input prompt.

Bonus tasks:

- [1 point] Using command option -p will display output in pages. User must press a button to display next page.
- [2 points] In paged output user can press a button to display previous pages.
- [1 point] Good comments or documentation in English language. – you are reading it
- [1 point] Correct handling of input files above 64 kB. – use file "big_file.txt" as input, there are words "ahoj" sprinkled among randomly generated lowercase letters.

Implementation

Argument input:

The program can be called using the following formula:

zadanie1 [-a stringA] [-b stringB] [-i inputFile] [-o outputFile] [-p] [-h]

[-a stringA]:

stringA is string that will be replaced in the input file

[-b stringB]:

stringB is string that will replace stringA in the input file

[-i inputFile]:

inputFile is the name of input file that will be processed.

[-o outputFile]:

inputFile is the name of input file that will be processed.

[-p]

Using this option will display output file in pages.

[-h]:

When this option is read program info is displayed and the following or previously read arguments are ignored.

This program can detect faulty input such as:

- arguments being provided again: *zadanie1 -a test1 -a test2*
- argument that provides input variable is called but that value is not provided: *zadanie1 -a*
- invalid argument: *zadanie1 -x*

Prompt input:

When stringA, string, inputFile or outputFile was not provided in arguments the user will be prompted to input the missing variables that are required to run the program. If user doesn't write anything and presses enter, they will be asked for the missing input again.

Function:

Algorithm works correctly if buffer size plus 2 is larger than length of stringA (string that will be replaced). If this condition is not met, stringA could be present in 3 buffers. This program assumes that stringA can be contained in 2 buffers at most. By default, buffer size is set to 100 and size of input strings is 32. End of buffer is determined by character $. If file was not opened correctly the user will be informed of this and the program will be terminated.

Program reads buffer from file and iterates through each byte and compares it to byte from stringA. If match is found the position of first matching symbol is saved and position in stringA is incremented. If all symbols are found and length of stringA is equal to the length of stringB then we return to the position of first matching symbol and simply replace letters of stringA with letters of string. Then we continue reading buffer until the end. When the entire buffer is checked we print the buffer, refill it with $, read the next buffer, and proceed until the end of file is reached.

Before:

1	s	t	R	A	2	3	4	5	$

After:

1	s	t	r	B	2	3	4	5	$

If we found some letters from stringA but before finding the whole string we find unmatching symbol, we return to the position of first found character from stringA and proceed checking for stringA from the following character again. The following example shows why need to do it in such a way, so we don't miss any matching strings. We will be replacing 'ababcd' with 'XXXXXX'.

Step 1:

1	a	b	a	b	A	b	C	d	$

Step 2:

1	a	b	a	b	A	b	C	D	$

Step 3:

1	a	b	X	X	X	X	X	X	$

If stringB is longer than stringA then we push the characters following found stringA to the right. Because of this fact buffer has reserved space for the maximum length difference of strings. Buffer always reads from file buffer_size bytes of characters but the real size of buffer is buffer_size + input_size bytes.

Before:

1	s	t	r	A	2	3	$	$	$

After:

1	s	s	s	t	r	B	2	3	$

If stringB is shorter than stringA then we push the characters following found stringA to the left. We also push the ending character $ to the left.

Before:

1	s	t	r	A	2	3	4	5	$

After:

1	s	B	2	3	4	5	$	5	$

Algorithm gets a lot more complex if stringA is found between 2 buffers (starts in one buffer and ends in the following buffer). If we are reading stringA and reach the end of buffer without finding the complete stringA, we copy buffer to buffer2 and read the next buffer. If we find the complete stringA we replace it with stringB, print buffer2 and proceed checking buffer.

Before:

1	2	3	4	5	6	7	s	t	$
r	A	8	9	10	11	12	13	14	$

After:

1	2	3	4	5	6	7	s	t	$
r	B	8	9	10	11	12	13	14	$

If we don't find the complete stringA, we return to the position of first found character of stringA and proceed checking until we read the entire buffer2, which will be printed. In the following example we are trying to replace 'ababcd' with 'XXXXXX'

Step 1:

1	2	3	4	5	6	a	b	a	$
b	a	b	c	d	7	8	9	10	$

Step 2:

1	2	3	4	5	6	a	b	a	$
b	a	b	c	d	7	8	9	10	$

Step 3:

1	2	3	4	5	6	a	b	X	$
X	X	X	X	X	7	8	9	10	$

If in this case stringB is longer than stringA we push the characters following stringA in buffer to the right.

Before:

1	2	3	4	5	6	7	s	t	$
r	A	8	9	10	$	$	$	$	$

After:

1	2	3	4	5	6	7	s	s	$
s	s	t	r	B	8	9	10	$	$

If in this case stringB is shorter than stringA we push the characters following stringA in buffer to the left. In the following example you can see how some of the characters that are being pushed to the left are pushed from buffer to buffer2.

Before:

1	2	3	s	s	s	s	s	t	$
r	A	4	5	6	7	8	9	10	$

After:

1	2	3	s	t	r	B	4	5	$
6	7	8	9	10	$	8	9	10	$

Macros:

Macros that are used in this program are implemented in separate file ‘zadanie1.txt’. There are number of macros that are called by the main program repeatedly. A good example of macro with arguments is clearBuffer, which takes address of buffer and its length as arguments. This macro fills the buffer with $ and is called before we read another buffer from file.