A picture containing circle

Description automatically generated**MARMARA UNIVERSITY**

**FACULTY OF ENGINEERING**

**COMPUTER ENGINEERING**

CSE3038

Computer Organization

Project1

Name Student Number

Talha SEZER 150119022

Sadık AKGEDİK 150119052

Mehmet Fatih ERDEM 150119054

Mustafa Tunahan BAŞ 150119055

Due Date : 19.04.2023 23.00

**1.Generate The Longest Palindrome**

This part of the program consists of 3 core procedures: longestPalindrome, loopStr and add\_to\_array.

The longestPalindrome procedure is responsible for looping the input string, checking the constraints, adding elements to our count array and generate strings. It is responsible for managing the core logic of the program

The loopStr procedure is responsible for looping the input string and doing the checks. It loads the current byte of the string to $t0 and checks if the current charater is null. If so it branches to the endloop. If it is not null, it starts checking. It loads the ASCII value of ‘A’ to $t1 and checks if the current byte is a letter or not. It branches to is\_letter procedure. The ASCII value of ‘z’ is loaded to $t2. Then, it branches to check\_case for checking if the letter is uppercase. If so, it converts it to lowercaseby branching to convert\_lower procedure. Once we get the lowercase letter we branch to add\_to\_array procedure. If the current byte is not a letter, the length counter and the string pointer is incremented in the not\_letter procedure.

The add\_to\_array multiplies the ASCII code by 4 to get the byte offset into the count array. Loads the count for the current character and increment the count. Lastly, it updates the count value in the array and save it.

The rest of the program implements an algorithm to generate the longest palindrome string. It loops from ‘a’ to ‘z’ and checks if the count of that array is odd. If so, it adds it to the mid string. If the count is even, it adds the half of it to the beg string. Lastly, the end string is generated by reversing the beg. However, this part of the program gives error and we weren’t able to solve that. So, we don’t call this program in the Main Menu. You can still examine the code in the file.

Graphical user interface, application

Description automatically generated

**2.Reverse Vowels**

This part of the program consists of 3 procedures: reverse\_vowels, read\_string and is\_vowel.

The is\_vowel procedure takes a one byte input(a character) in register $a0. It works by iterating through a global array of lowercase vowels, and comparing the character to them. In each iteration, the input is first compared to the lowercase character, and if it doesn’t match then it is compared to the uppercase version of that character. This way, the character is compared to each vowel in the English alphabet (if there’s a match the procedure will return early). If the character is a vowel, 1 is returned in $v0, otherwise 0 is returned.

The read\_string procedure takes no arguments. It first creates a block of buffer\_size bytes(a global variable) on the heap, takes a user input and places it in the allocated memory. Then, it traverses the user input string, it calls the is\_vowel procedure for each character and when a character is vowel it pushes that character on the stack. The iteration stops when a new line character or null character is found (the new line character is overridden with null character). After the iteration, the sp is restored to its original state and the address of the user input is returned in $v0, and the number of vowels in the user input is returned in $v1.

The reverse\_vowels procedure first calls the read\_string and relocates the stack pointer to show the address of the last vowel pushed in the read\_string procedure (This is done by decrementing the sp by $v1 + 32 (because read\_string procedure saves 8 registers on the stack)). Then the input string returned from the read\_string procedure is traversed and each character is checked if it’s vowel again. If a character is not a vowel then it is printed on the screen. However, when a vowel is encountered, it pops the corresponding vowel on the stack and prints it instead. After the loop, a new line character is printed and the sp is restored to its original position.

**Graphical user interface, text, application, email

Description automatically generatedSample Run**

**3. Square Free Number**

This part of the program there are 4 procedures. Find\_prime\_numbers, is\_square\_free, is\_not\_square\_free, find\_prime\_for\_print.

In find\_prime\_numbers, the input number is divided by the smallest prime number which is 2. If remainder is 0, the number of prime factor counter is increased by 1. When input can divide by the current prime number, input will be reassign to divided value. We are holding a prime counter which holds the value of how many times the current prime number is used. If this counter’s value become 2, we can say that the input is not a square free number and stop the loop. If modified input become 1, we can say that input is square free.

In is\_not\_square\_free, first we print the input, than we print “is not a square-free number”. Finally we terminate the program.

In is\_square\_free, first we print the input number, than we print “is a square-free number and has”. After that we print the number of prime factors that we find in find\_prime\_numbers. Than we print “distinct prime factors: ”. Than we call the find\_prime\_for\_print.

In find\_prime\_for\_print, we divide the input number by 2. If remainder is not zero, we increment prime number. If it is zero we print the current prime number.

**Output:**

Text

Description automatically generated with low confidence