Unicode Converter and Translator Analysis Writeup

CSARCH2 S12 Group 9

In developing the Unicode Translator, we encountered challenges in translating various UTF inputs to Unicode hexadecimal. Python simplified the conversion from Unicode hexadecimal to the three UTF formats. However, we spent some time figuring out the reverse process. We encountered exceptions when working with Python's built-in encode and decode functions. After several tests, we realized the necessity of writing the utf_to_hex() function, as seen in the code snippet below.

- bytes.fromhex(): Converts a hexadecimal string to bytes format.
- decode(): Decodes the bytes using UTF encoding to obtain a UTF string.
- ord(): Converts the first character of the UTF string to its Unicode code point.
- hex(): Converts the Unicode code point to its hexadecimal representation.
- upper(): Converts the hexadecimal string to uppercase.

For UTF-16 and UTF-32, we had to specify the use of big endian by appending the characters 'be' to the decode() string parameter. It took us a while to realize that without using ord(), the translation would not work properly.

```
def utf_to_hex(hex_input, conversion_type):
    try:
        if conversion_type == "Utf-8":
            hex_result = hex(ord(bytes.fromhex(hex_input).decode('utf-8'))).upper()
        elif conversion_type == "Utf-16":
            # Split the hex string into bytes
            bytes_array = bytes.fromhex(hex_input)
            # Decode bytes as UTF-16
            utf16_string = bytes_array.decode('utf-16be')
            # Convert the first character of the decoded string to hex
            hex_result = hex(ord(utf16_string[0])).upper()
        elif conversion_type == "Utf-32":
            hex_result = hex(ord(bytes.fromhex(hex_input).decode('utf-32be'))).upper()

        return hex_result
    except ValueError:
        return None
```

In addition to the translation section of the code, we faced challenges with input validation, particularly regarding invalid hexadecimal digits and input ranges for Unicode and UTF strings. We developed two functions for input validation, namely validate_hex_input and validate_utf_input.

The validate_hex_input() function handles empty string inputs, invalid hexadecimal digits, and out-of-range inputs. We also accounted for the possibility of the user starting the input with '0x'. We set the maximum Unicode hexadecimal input to 0x10FFFF, as the code will automatically convert to all three UTF formats.

```
def validate_hex_input(text):

    if text == '':
        return False  # Do not allow empty string
    if text.startswith('0x') or text.startswith('0X'):
        text = text[2:]
    if all(c.upper() in '0123456789ABCDEF' for c in text):
        hex_value = int(text, 16)
    else:
        return False
    if  hex_value <= 0x10FFFF and len(text) <= 8:
        return True
    else:
        return False  # Invalid encoding specified
```

The validate_utf_input() function handles empty string inputs, invalid hexadecimal digits, invalid input string lengths, and out-of-range inputs. For UTF-8, we allow inputs with less than 4 bytes (8 hexadecimal digits). However, for UTF-16 and UTF-32, we require an input string length of 8 characters. We implemented a special check for UTF-16 inputs to ensure adherence to the "DxxxDxxx" format. Additionally, we verify if the range of each half falls within D800 to DFFF.

```
def validate_utf_input(text):
    if text == '':
        return False  # Do not allow empty string
    if conversion_var.get() == "Utf-8":
        if  all(c.upper() in '0123456789ABCDEF' for c in text) and len(text) <= 8:
            return True
        return False
    elif conversion_var.get() == "Utf-16":
        if  check_utf16_format(text) and all(c.upper() in '0123456789ABCDEF' for c in text) and len(text) == 8:
            return True
        return False
    elif conversion_var.get() == "Utf-32":
        if  all(c.upper() in '0123456789ABCDEF' for c in text) and len(text) == 8:
            return True
        return False
    else:
        return False
```

```
def check_utf16_format(input_string):
    # Define the regular expression pattern
    pattern = re.compile(r'^D[0-9A-Fa-f]{3}D[0-9A-Fa-f]{3}$')

    # Check if the input string matches the pattern
    if pattern.match(input_string):
        return True
    else:
        return False
```