Elementary Systems Programming Project

<ProcDoc>

01286120 Elementary Systems Programming

Software Engineering Program

Faculty of Engineering, KMITL

By

66010991 Diyaan Pulikkal

C1 Document Conversion Tool – ProcDoc

Overview

This program allows user to convert an individual file to a specified file format. The possible conversion formats are provided in the form of table that will be depicted shortly after this text section. This program also provides the functionality of copying a certain file to a specified directory. In big picture, the process runs in this order: [**Command line argument parsing** => **Conversion info construction** => **Matching info**=> **Conversion process**]. It is recommended to read this document from top to bottom to achieve a coherent interpretation of this program.

Possible conversion list:

From:	To:
csv	html
json	xml
pdf (text-only)	txt
txt	pdf
txt	docx
xml	json

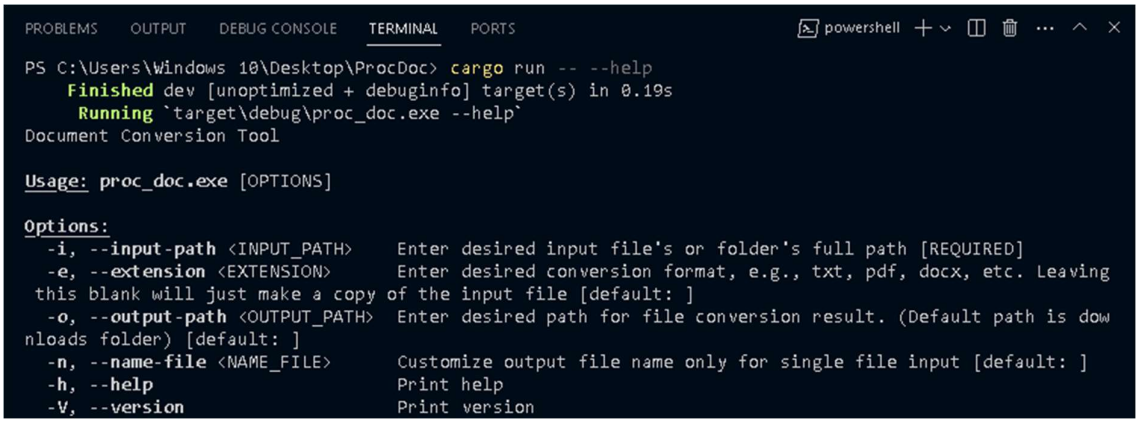
Command Line Argument Parsing

Here is the parser struct created using the crate **`clap`** (v4.4.7):

```rust
#[derive(Parser)]
#[command(author="Diyaan Pulikkal <66010991@kmitl.ac.th>", version="0.1.0", about="Document Conversion Tool", long_about = None)]
4 implementations
pub struct Instruction {
    /// Enter desired input file's full path [REQUIRED]
    #[arg(long, short)]
    pub input_path: Option<String>,

    /// Enter desired conversion format, e.g., txt, pdf, docx, etc. Leaving this blank will just make a copy of the input file.
    #[arg(long, short, default_value_t = String::from(""))]
    extension: String,

    /// Enter desired path for file conversion result. (Default path is downloads folder)
    #[arg(long, short, default_value_t = String::from(""))]
    output_path: String,

    /// Customize output file name only for single file input
    #[arg(long, short, default_value_t = String::from(""))]
    name_file: String,
}
```

The **`Instruction::parse()`** is initialized in the **`main()`** function of main.rs.

By entering `cargo run -- --help` into the terminal:



The only required argument for this program is the input file's path. The rest are dependent on how one would want the utilize this program. The purpose of each argument options is stated in the image.

Conversion Info Construction

The objective of this progression is to create a grouped pieces of information readable by the latter part of the program. This could be accomplished by filling the `ConversionInfo` struct using `ConversionInfo::new()` function:

```rust
impl ConversionInfo {
    fn new(
        input_path: String,
        input_extension: String,
        output_path: String,
        output_extension: String,
        name_file: String,
    ) -> ConversionInfo {
        return ConversionInfo {
            input_path,
            input_extension,
            output_path,
            output_extension,
            name_file,
        };
    }
}
```

```rust
struct ConversionInfo {
    input_path: String,
    input_extension: String,
    output_path: String,
    output_extension: String,
    name_file: String,
}
```

This whole process is done in the `verify_args()` function. The function attempts to acknowledge the extension of the input file without the need for user's provision.

```rust
    let file_path: &Path = Path::new(&input_path);
    if let Some(ext: &OsStr) = file_path.extension() {
        if let Some(ext_str: &str) = ext.to_str() {
            if output_extension == "".to_string() {
                output_extension = ext_str.to_string();
            }
            input_extension = ext_str.to_string().clone();
        }
    }
```

For now, ignore the conditional statement that that checks `output_extension`.

The upcoming set of subsections demonstrates argument errors handling. The following images are captured from **`verify_args()`** in lib.rs except for Error 1**.** The program checks for these argument errors in order. If one error is detected, the program will handle it without having to examine the rest of the argument.

Error 1: No input file provided.

```rust
if instruction.input_path.is_none(){
    eprintln!("Please enter the path of input file.");
    std::process::exit(code: 0);
}
```

This code snippet resides in the **`main()`** function within the main.rs file. It checks if the **`input_path`** given by the user is a **`None`** variant of **`Option<String>.`** If so, the displayed **`eprintln!()`** macro is executed and the program exits.

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\Windows 10\Desktop\ProcDoc> cargo run --
    Finished dev [unoptimized + debuginfo] target(s) in 0.19s
     Running `target\debug\proc_doc.exe`
Please enter the path of input file.
PS C:\Users\Windows 10\Desktop\ProcDoc>
```

Error 2: Non-existent input file

```rust
if fs::metadata(&input_path).is_err() {
    eprintln!("Input file not found.");
    exit(code: 0);
}
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\Windows 10\Desktop\ProcDoc> cargo run -- -i randomtext
    Finished dev [unoptimized + debuginfo] target(s) in 0.19s
     Running `target\debug\proc_doc.exe -i randomtext`
Input file not found.
PS C:\Users\Windows 10\Desktop\ProcDoc>
```

The program terminates if the **`metadata()`** function returns **`Err`** variant.

Error 3: Non-existent output folder.

```rust
if output_path != "".to_string() && fs::metadata(&output_path).is_err() {
    eprintln!("Output folder not found.");
    exit(code: 0);
}
```

If the **`output_path`** argument retrieved is not the default “” value (which means the user has entered something) and the **`metadata()`** function of the **`output_path`** reference returns an **`Err`** variant, the program prints the error message and terminates.

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\Windows 10\Desktop\ProcDoc> cargo run -- -i "test_files/txt.txt" -o randomtext
   Compiling proc_doc v0.1.0 (C:\Users\Windows 10\Desktop\ProcDoc)
    Finished dev [unoptimized + debuginfo] target(s) in 2.22s
     Running `target\debug\proc_doc.exe -i test_files/txt.txt -o randomtext`
Output folder not found.
PS C:\Users\Windows 10\Desktop\ProcDoc>
```

Error 4: No output folder provided.

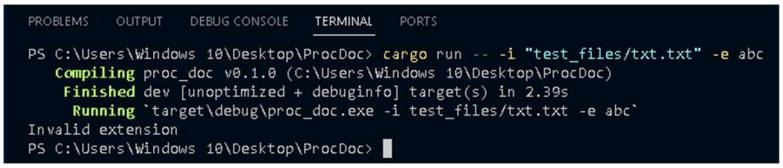
If the **`output_path`** is a default value (“”), the program uses the function **`download_dir()`** function from the crate **`dirs`** to search for the system’s default Downloads directory, convert it to **`String`** and assign it as the **`output_path`**. If the **`Some`** variant of the Downloads directory could not be initialized through **`if let`** structure, which means the crate **`dirs`** somehow could not find Downloads folder, then the program prints the error message and then exits. The device that documented this had a Downloads folder so it was difficult to show error handling.

Error 5: No output extension provided

When the program was figuring out the file format of the input file, it also checks if the user has provided the output extension by seeing if it is “”. If so, the program assigns **`output_extension`** as the same value as **`input_extension`**.

Error 6: Unsupported input/output extension

`avail_ext` contains an array of available conversion format. An **`if`** statement was used to check if the array contains both **`output_extension`** and **`input_extension`**. If not, then the error message is displayed and the program exits. Then, **`avail_text`** is dropped to (insignificantly) improve performance.



Extra: Output file naming

```
if name_file == "".to_string() {
    let path_for_name: &Path = Path::new(&input_path);
    if let Some(path_name: &OsStr) = path_for_name.file_stem() {
        if let Some(str_name: &str) = path_name.to_str() {
            if input_extension == output_extension {
                name_file = format!("{}-copied", str_name.to_string());
            } else {
                name_file = format!("{}-converted", str_name.to_string());
            }
        }
    }
}
```

It is not compulsory for the user to name output file. In fact, it is purely for the pleasure of customization. **If no `name_file` was provided**, the program gets the input file name, excluding extension, using **`file_stem()`** method from **`std::path::Path`**. The program then checks if the input and output extensions are the same. If so, the output file name will be “inputfilename-copied”. If the extensions are different, the out the output file name will be “inputfilename-converted”.

Construction and continuation:

Now the information is finalized and ready to be directed to its respective conversion function through the **`direct_info()`** function.

```
let finalized_info: ConversionInfo = ConversionInfo::new(
    input_path,
    input_extension,
    output_path,
    output_extension,
    name_file,
);

if let Err(e: Box<dyn Error>) = direct_info(finalized_info) {
    eprintln!("{}", e);
}
} fn verify_args
```

Matching Info

In this `direct_info()` function:

```rust
// Direct the processed argument to their respective conversion function
fn direct_info(info: ConversionInfo) -> MyResultBox {
    if info.input_extension == info.output_extension {
        if let Err(e: Box<dyn Error>) = duplicate_file(info) {
            eprintln!("{}", e)
        }
        return Ok(());
    }

    match (
        info.input_extension.as_str(),
        info.output_extension.as_str(),
    ) {
        ("txt", "pdf") => txt_to_pdf(info),
        ("pdf", "txt") => pdf_to_txt(info),
        ("json", "xml") => json_to_xml(info),
        ("xml", "json") => xml_to_json(info),
        ("csv", "html") => csv_to_html(info),
        ("txt", "docx") => txt_to_docx(info),
        _ => Ok(inconvertible_formats(info)),
    }
}
```

The function checks if the `input_extension` and the `output_extension` is equivalent. If the result is positive, the program calls `duplicate_file()` and `returns Ok` variant. Else, the subsequent `match` expression calls an appropriate function respective to the extensions. If the extensions are not in the available conversion pairs, the `inconvertible_formats()` function is called to show users the conversion functions:

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS

PS C:\Users\Windows 10\Desktop\ProcDoc> cargo run -- -i "test_files/xml.xml" -e docx
   Compiling proc_doc v0.1.0 (C:\Users\Windows 10\Desktop\ProcDoc)
    Finished dev [unoptimized + debuginfo] target(s) in 2.03s
     Running `target\debug\proc_doc.exe -i test_files/xml.xml -e docx`
Cannot convert xml to docx
See possible conversions:
txt => pdf
txt => docx
pdf(text only) => txt
xml <=> json
csv => html
PS C:\Users\Windows 10\Desktop\ProcDoc>
```

Conversion Process

Function 1: Duplicate file

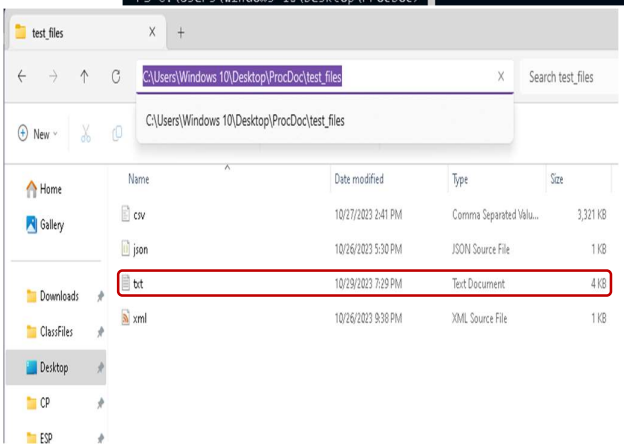
Explanation:

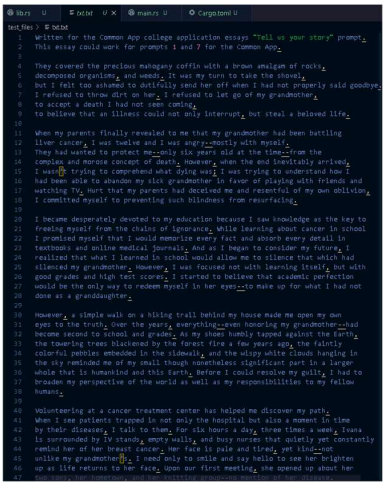
```rust
fn duplicate_file(info: ConversionInfo) -> MyResultBox {
    let out_str_path: String = format!(
        "{}/{}.{}",
        info.output_path, info.name_file, info.output_extension
    );
    fs::copy(from: info.input_path, to: out_str_path)?;

    Ok(())
}
```

This a basic function that just uses **`fs::copy()`** function to duplicate the same file to the output folder (Downloads folder by default).

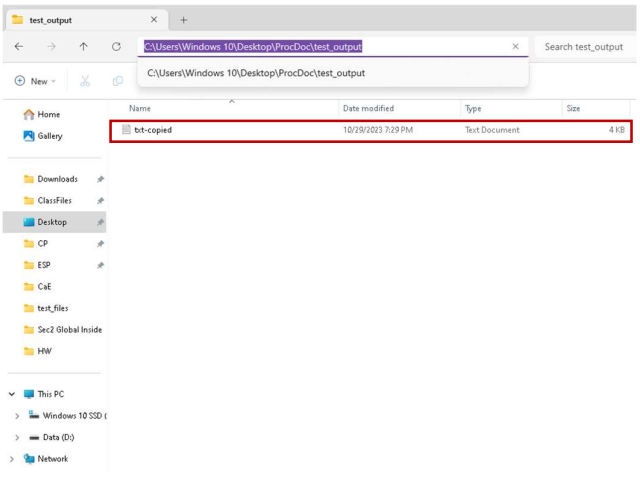
Input:

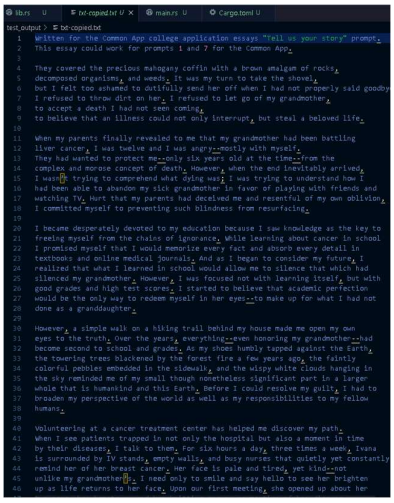
```
PS C:\Users\Windows 10\Desktop\ProcDoc> cargo run -- -i "test_files/txt.txt" -o "test_output"
    Finished dev [unoptimized + debuginfo] target(s) in 0.19s
     Running `target\debug\proc_doc.exe -i test_files/txt.txt -o test_output`
Success!
PS C:\Users\Windows 10\Desktop\ProcDoc>
```





Output:





Function 2: txt to pdf

Explanation:

```
use printpdf::*;
```

This function mainly utilizes the crate **`printpdf`** v0.6.0.

```
fn txt_to_pdf(info: ConversionInfo) -> MyResultBox {
    // Get String from txt file
    let mut text = String::new();
    File::open(info.input_path)?.read_to_string(&mut text)?;

    // Create a document object
    let (doc, page1, layer1) = PdfDocument::new("", Mm(210.0), Mm(297.0), "Layer 1");
    let font = doc.add_builtin_font(BuiltinFont::TimesRoman)?;
    let mut layer = doc.get_page(page1).get_layer(layer1);
```

This function starts off by acquiring **`String`** object from the **`input_path`** using the basic **`File::open().read_to_string()`** method. Then, we create a **`PdfDocument`** object provided by the **`printpdf`** crate. The parameter of the new() function is **`(Document title, First page width, First page height, starting Layer).`** Times Roman is selected as the initial font because of its commonness. The **`layer`** variable is declared to be the first layer of **`page1`** of **`doc`**.

```
    // Divide the text into lines with 110 characters each
    let text_char: Vec<char> = text.chars().collect();
    let char_per_lines: usize = 110;
    let no_of_lines: usize = text_char.len() / char_per_lines;
    let mut count: f32 = 0.0;
```

A few significant variables are needed during the upcoming **`for`** loop. **`char_per_lines`** is the amount of text characters that will be in each line of the pdf. An amount of 110 characters per line is chosen because of its suitability when utilized on common text where there are both capital and lowercase characters in a sensible ratio. This program is not suitable for txt file that only contain capital letters because a portion of characters will trail off the page. **`no_of_lines`** is self-explanatory; it holds the number of lines of text that will be printed into the pdf. Since **`usize`** divided by **`usize`** cannot result in a float type, it is basically floor division. **`count`** will be used to count the number of lines printed into the **`layer`** of **`doc`** during the **`for`** loop iteration. See next page…

```
for i in 0..no_of_lines {
    layer.use_text(
        text.slice((char_per_lines * i)..(char_per_lines * (i + 1))),
        12.0,
        Mm(10.0),
        Mm(287.0 - (10.0 * count)),
        &font,
    );
    if i == no_of_lines - 1 {
        layer.use_text(
            text.slice((char_per_lines * (i + 1))..),
            12.0,
            Mm(10.0),
            Mm(287.0 - (10.0 * (count + 1.0))),
            &font,
        );
    }
    // If a page already contains 27 lines, a new page begins
    if count == 27.0 {
        let (page1, layer1) = doc.add_page(Mm(210.0), Mm(297.0), "Layer 1");
        layer = doc.get_page(page1).get_layer(layer1);
        count = 0.0;
    } else {
        count = count + 1.0;
    }
}
```

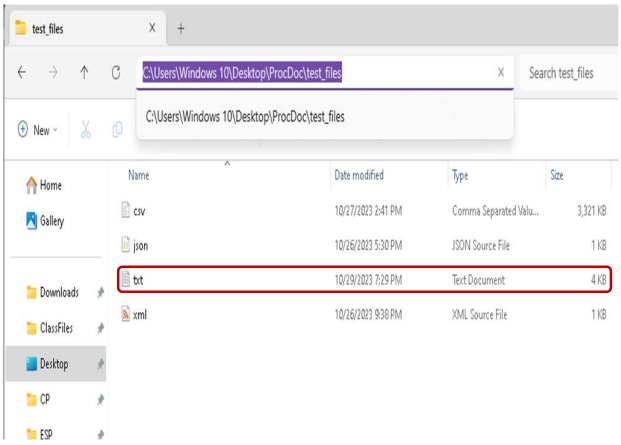
The number of times that this loop will run equates to the value of **`no_of_line`**, which means the loop will print the text into the pdf line by line. In the first run, with the use of **`slicestring`** crate, the method **`slice()`** is implemented to take a copy of the first 110 characters off the **`text`** content. The sliced text is then put in the **`layer.use_text()`** method to apply the text to the layer of the document object. The value **`12.0`** is the font size that will be applied in the pdf. The **`Mm(10.0)`** is the x-position of where the text will start in the page while the **`Mm(287.0 – (10.0 * (count + 1.0)))`** is the y-position. The value of y-position is in that way because the y value needs to change after each line so the text does not cover each other by having the same y position. The second **`if`** statement checks if the iteration is at the last line of the **`text`**. This is required because it is likely that the exact number of characters is not a multiple of 110. In the last line, there would definitely be less than 110 characters. Without any concern, the program slices the rest of the **`text`** and use it similarly in the **`use_text()`** method. Next, the program checks **`if`** the value of **`count`** is equal to 27 lines. 27 is observed to be the most appropriate maximum number of lines in a page based on trials and errors. If the condition is met, **`doc.add_page()`** is called to add a new page, **`page1`** (lack of name ideas), with a default layer, **`layer1`**. Additionally, the **`count`** value is also reset to **`0.0`**. For the case of **`count`** not being **`27.0`** yet, count's value is incremented by **`1.0`** and the iteration goes on.

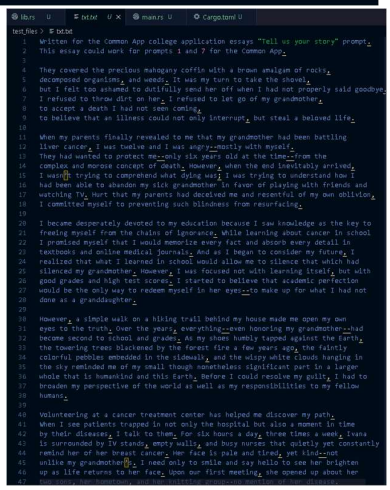
```rust
    // Create the new pdf file and apply the object created
    let file: File = File::create(path: format!(
        "{}/{}.{}",
        info.output_path, info.name_file, info.output_extension
    ))?;
    let mut file: BufWriter<File> = BufWriter::new(inner: file);
    doc.save(target: &mut file)?;

    Ok(())
} fn txt_to_pdf
```

The final part of the code is to **`save()`** the doc object to the actual pdf file. **`BufWriter`** of the created output file is initialized to be able to use **`save()`** method provided by **`printpdf.`**

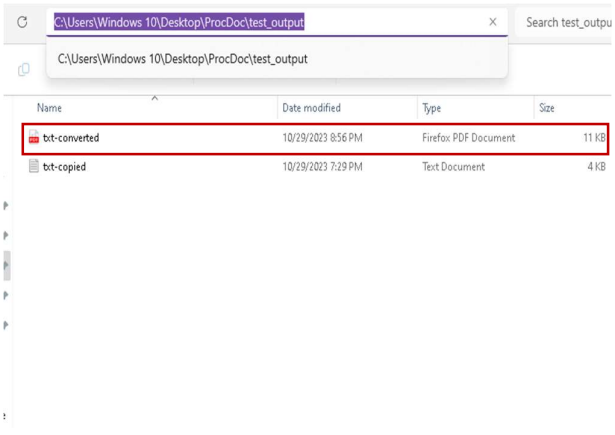
Input:

```
PS C:\Users\Windows 10\Desktop\ProcDoc> cargo run -- -i "test_files/txt.txt" -o "test_output" -e pdf
   Compiling proc_doc v0.1.0 (C:\Users\Windows 10\Desktop\ProcDoc)
    Finished dev [unoptimized + debuginfo] target(s) in 2.89s
     Running `target\debug\proc_doc.exe -i test_files/txt.txt -o test_output -e pdf`
Success!
```





Output:



Function 3: pdf to txt

Explanation:

```
use pdf_extract::extract_text;
```

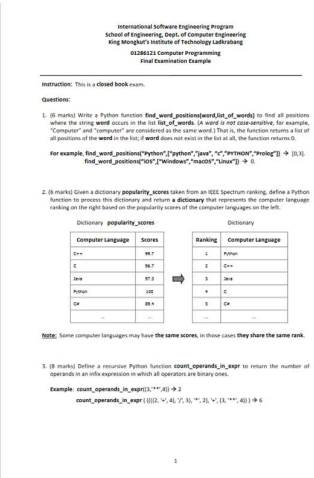
The crate **`pdf_extract`** v0.7.2 is used.

```
fn pdf_to_txt(info: ConversionInfo) -> MyResultBox {
    let text: String = extract_text(&info.input_path).unwrap_or(default: "".to_string());

    let mut out_file: File = File::create(path: format!(
        "{}/{}.{}",
        info.output_path, info.name_file, info.output_extension
    ))?;

    out_file.write_all(buf: text.as_bytes())?;

    Ok(())
}
```

This is a simple one. All the **`text`** content in the pdf is extracted using the provided method **`extract_text()`**. This will ignore all image contents within the pdf so the output might not satisfy the need of users with pdf containing graphics. However, it is safe to use even with pdf that contains images since the process does not affect the input file. Lastly, the text content is then written using **`write_all()`** method provided by **`std::io::Write.`**

Input:

```
PS C:\Users\Windows 10\Desktop\ProcDoc> cargo run -- -i "C:\Users\Windows 10\Downloads\Python Final Exam Example 2022.pdf" -e txt
    Finished dev [unoptimized + debuginfo] target(s) in 0.20s
     Running `target\debug\proc_doc.exe -i "C:\Users\Windows 10\Downloads\Python Final Exam Example 2022.pdf" -e txt`
Success!
PS C:\Users\Windows 10\Desktop\ProcDoc>
```



International Software Engineering Program
School of Engineering, Dept. of computer Engineering
King Mongkut's Institute of Technology Ladkrabang

01286121 Computer Programming
Final Examination Example

Instruction: This is a closed book exam.

Questions:

1. (6 marks) Write a Python function **find_word_positions(word,list_of_words)** to find all positions where the string **word** occurs in the list **list_of_words**. (*A word is not case-sensitive*, for example, "Computer" and "computer" are considered as the same word.) That is, the function returns a list of all positions of the **word** in the list; if **word** does not exist in the list at all, the function returns 0.

For example, **find_word_positions("Python",["python","java","c","PYTHON","Prolog"])** → [0,3].
find_word_positions("iOS",["Windows","macOS","Linux"]) → 0.

2. (6 marks) Given a dictionary **popularity_scores** taken from an IEEE Spectrum ranking, define a Python function to process this dictionary and return **a dictionary** that represents the computer language ranking on the right based on the popularity scores of the computer languages on the left.

Dictionary **popularity_scores**

Computer Language	Scores
C++	99.7
C	96.7
Java	97.3
Python	100
C#	89.4
...	...

Dictionary

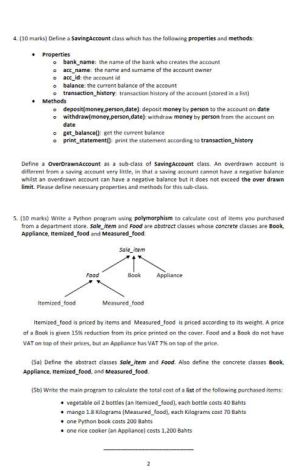
Ranking	Computer Language
1	Python
2	C++
3	Java
4	C
5	C#
...	...

Note: Some computer languages may have **the same scores**, in those cases **they share the same rank**.

3. (8 marks) Define a recursive Python function **count_operands_in_expr** to return the number of operands in an infix expression in which all operators are binary ones.

Example: **count_operands_in_expr**([3,'**',4]) → 2
count_operands_in_expr ([[[2, '+', 4], '/', 3], '**', 2], '+', [3, '**', 4]]) → 6

1



4. (10 marks) Define a **SavingAccount** class which has the following **properties** and **methods**:

- **Properties**
  - **bank_name**: the name of the bank who creates the account
  - **acc_name**: the name and surname of the account owner
  - **acc_id**: the account id
  - **balance**: the current balance of the account
  - **transaction_history**: transaction history of the account (stored in a list)
- **Methods**
  - **deposit(money,person,date)**: deposit **money** by **person** to the account on **date**
  - **withdraw(money,person,date)**: withdraw **money** by **person** from the account on **date**
  - **get_balance()**: get the current balance
  - **print_statement()**: print the statement according to **transaction_history**

Define a **OverDrawnAccount** as a sub-class of **SavingAccount** class. An overdrawn account is different from a saving account very little, in that a saving account cannot have a negative balance whilst an overdrawn account can have a negative balance but it does not exceed **the over drawn limit**. Please define necessary properties and methods for this sub-class.

5. (10 marks) Write a Python program using **polymorphism** to calculate cost of items you purchased from a department store. ***Sale_item*** and ***Food*** are *abstract* classes whose *concrete* classes are **Book, Appliance, Itemized_food** and **Measured_food**.

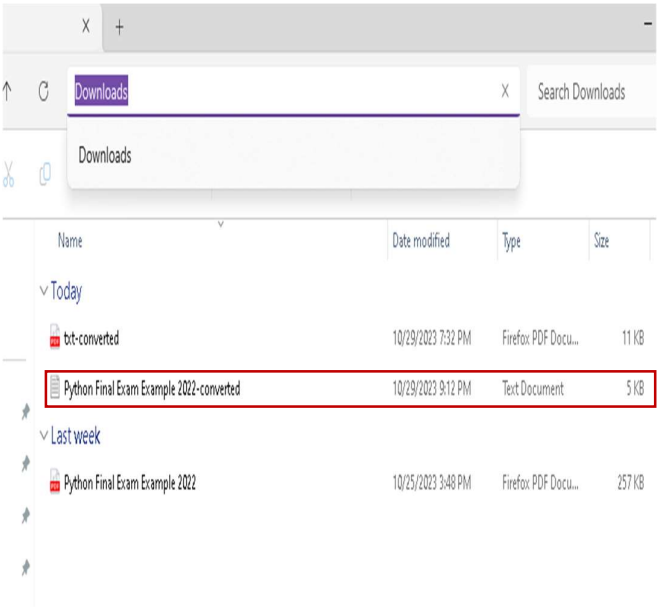
Itemized_food is priced by items and Measured_food is priced according to its weight. A price of a Book is given 15% reduction from its price printed on the cover. Food and a Book do not have VAT on top of their prices, but an Appliance has VAT 7% on top of the price.

(5a) Define the abstract classes ***Sale_item*** and ***Food***. Also define the concrete classes **Book, Appliance, Itemized_food**, and **Measured_food**.

(5b) Write the main program to calculate the total cost of a **list** of the following purchased items:

- vegetable oil 2 bottles (an Itemized_food), each bottle costs 40 Bahts
- mango 1.8 Kilograms (Measured_food), each Kilograms cost 70 Bahts
- one Python book costs 200 Bahts
- one rice cooker (an Appliance) costs 1,200 Bahts

2



Output:

1

International Software Engineering Program
School of Engineering, Dept. of Computer Engineering
King Mongkut's Institute of Technology Ladkrabang

01286121 Computer Programming
Final Examination Example

Instruction: This is a closed book exam.

Questions:

1. (6 marks) Write a Python function find_word_positions(word,list_of_words) to find all position where the string word occurs in the list list_of_words. (A word is not case-sensitive, for example "Computer" and "computer" are considered as the same word.) That is, the function returns a list of all positions of the word in the list; if word does not exist in the list at all, the function returns 0.
For example, find_word_positions("Python",["python","java", "c","PYTHON","Prolog"]) → [0,3].
find_word_positions("iOS",["Windows","macOS","Linux"]) → 0.

2. (6 marks) Given a dictionary popularity_scores taken from an IEEE Spectrum ranking, define a Python function to process this dictionary and return a dictionary that represents the computer language ranking on the right based on the popularity scores of the computer languages on the left.

Dictionary popularity_scores Dictionary

Computer Language	Scores	Ranking	Computer Language
C++	99.7	1	Python
C	96.7	2	C++
Java	97.5	3	Java
Python	100	4	C
C#	89.4	5	C#
…	…	…	…

Note: Some computer languages may have the same scores, in those cases they share the same rank.

3. (8 marks) Define a recursive Python function count_operands_in_expr to return the number of operands in an infix expression in which all operators are binary ones.
Example: count_operands_in_expr((3,'**',4)) → 2

count_operands_in_expr (((((2, '+', 4), '/', 3), '*', 2), '+', (3, '**', 4))) → 6

2

4. (10 marks) Define a SavingAccount class which has the following properties and methods:
- Properties
  - bank_name: the name of the bank who creates the account
  - acc_name: the name and surname of the account owner
  - acc_id: the account id
  - balance: the current balance of the account
  - transaction_history: transaction history of the account (stored in a list)
- Methods
  - deposit(money,person,date): deposit money by person to the account on date
  - withdraw(money,person,date): withdraw money by person from the account on date
  - get_balance(): get the current balance
  - print_statement(): print the statement according to transaction_history

Define a OverDrawnAccount as a sub-class of SavingAccount class. An overdrawn account is different from a saving account very little, in that a saving account cannot have a negative balance whilst an overdrawn account can have a negative balance but it does not exceed the over drawn limit. Please define necessary properties and methods for this sub-class.

5. (10 marks) Write a Python program using polymorphism to calculate cost of items you purchas from a department store. Sale_item and Food are abstract classes whose concrete classes are Book, Appliance, Itemized_food and Measured_food.

Sale_item

Food Book Appliance

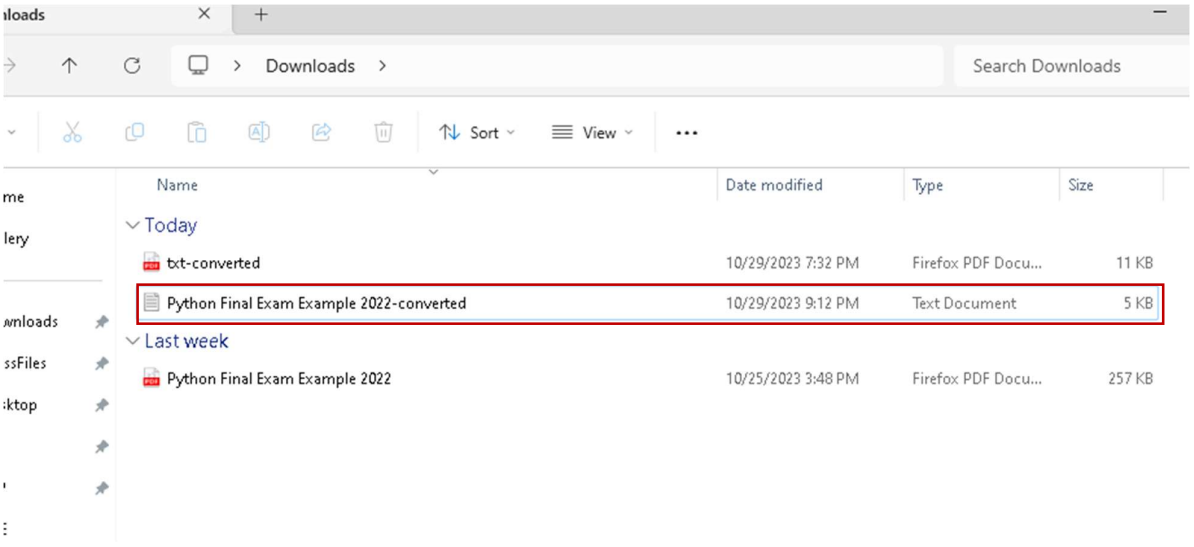
Itemized_food Measured_food

Itemized_food is priced by items and Measured_food is priced according to its weight. A price of a Book is given 15% reduction from its price printed on the cover. Food and a Book do not have VAT on top of their prices, but an Appliance has VAT 7% on top of the price.

(5a) Define the abstract classes Sale_item and Food. Also define the concrete classes Book, Appliance, Itemized_food, and Measured_food.

(5b) Write the main program to calculate the total cost of a list of the following purchased items:

- vegetable oil 2 bottles (an Itemized_food), each bottle costs 40 Bahts
- mango 1.8 Kilograms (Measured_food), each Kilograms cost 70 Bahts
- one Python book costs 200 Bahts
- one rice cooker (an Appliance) costs 1,200 Bahts



Function 4: json to xml

Explanation:

```rust
use serde_json::{to_string_pretty, Value};
use serde_xml_rs::to_string;
```

This function mainly uses **`serde_jason`** and **`serde_xml_rs`** crates to manipulate both the data formats.

```rust
fn json_to_xml(info: ConversionInfo) -> MyResultBox {
    let mut json_content: String = String::new();
    File::open(info.input_path)?.read_to_string(buf: &mut json_content)?;

    let json_data: Value = serde_json::from_str(&json_content)?;
    let xml_content: String = to_string(&json_data)?;
```

The program reads the json content from the **`input_path`**.
`serde_json::from_str()` is then used to deserialize the data to type **`Value`**.
Moving forward, the function **`to_string()`** from **`serde_xml_rs`** is called on **`data`** object to serialize it to **`String.`**

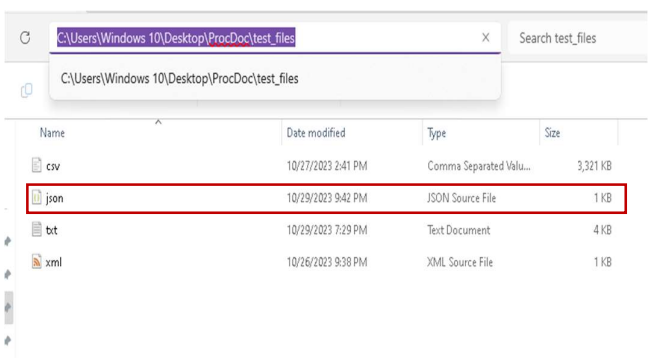
```rust
    let mut xml_file: File = File::create(path: format!(
        "{}/{}.{}",
        info.output_path, info.name_file, info.output_extension
    ))?;
    xml_file.write_all(buf: xml_content.as_bytes())?;

    Ok(())
```

Finally, the xml file is created and written using **`std::fs`** and **`std::io::Write.`**

Input:

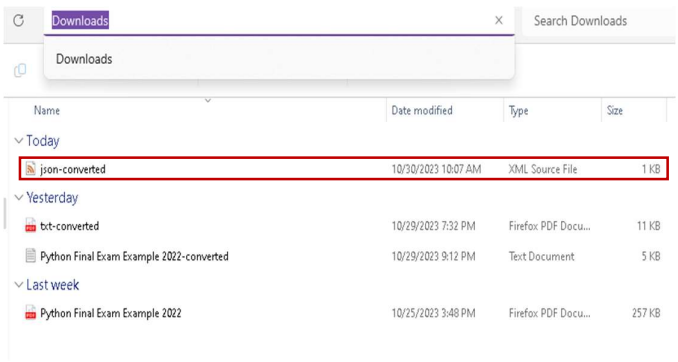
```json
{
    "quiz": {
        "sport": {
            "q1": {
                "question": "Which one is correct team name in NBA?",
                "options": [
                    "New York Bulls",
                    "Los Angeles Kings",
                    "Golden State Warriros",
                    "Huston Rocket"
                ],
                "answer": "Huston Rocket"
            }
        },
        "maths": {
            "q1": {
                "question": "5 + 7 = ?",
                "options": [
                    "10",
                    "11",
                    "12",
                    "13"
                ],
                "answer": "12"
            },
            "q2": {
                "question": "12 - 8 = ?",
                "options": [
                    "1",
                    "2",
                    "3",
                    "4"
                ],
                "answer": "4"
            }
        }
    }
}
```

```
PS C:\Users\Windows 10\Desktop\ProcDoc> cargo run -- -i "test_files/json.json" -e xml
    Finished dev [unoptimized + debuginfo] target(s) in 0.22s
     Running `target\debug\proc_doc.exe -i test_files/json.json -e xml`
Success!
PS C:\Users\Windows 10\Desktop\ProcDoc>
```



Output:

```xml
<?xml version="1.0" encoding="utf-8"?>
<quiz>
    <maths>
        <q1>
            <answer>12</answer>
            <options>10</options>
            <options>11</options>
            <options>12</options>
            <options>13</options>
            <question>5 + 7 = ?</question>
        </q1>
        <q2>
            <answer>4</answer>
            <options>1</options>
            <options>2</options>
            <options>3</options>
            <options>4</options>
            <question>12 - 8 = ?</question>
        </q2>
    </maths>
    <sport>
        <q1>
            <answer>Huston Rocket</answer>
            <options>New York Bulls</options>
            <options>Los Angeles Kings</options>
            <options>Golden State Warriros</options>
            <options>Huston Rocket</options>
            <question>Which one is correct team name in NBA?</question>
        </q1>
    </sport>
</quiz>
```



The real output will have all xml content written in a span of a line. This image was already formatted using VScode.

Function 5: xml to json

Explanation:

```rust
use quickxml_to_serde::{xml_string_to_json, Config};
```

This crate uses the **`quickxml_to_serde`** v0.5.0.

```rust
fn xml_to_json(info: ConversionInfo) -> MyResultBox {
    let mut xml_data: String = String::new();
    File::open(info.input_path)?.read_to_string(buf: &mut xml_data)?;

    let conf = Config::new_with_custom_values(true, "", "txt", NullValue::Null);
    let json = xml_string_to_json(xml_data.to_owned(), &conf);
```

The **`xml_data`** from the file is read using **`std::fs`**. A new variable **`conf`** is assigned with **`Config`** object, which holds the information on how to handle the conversion type:

```rust
pub struct Config {
    pub leading_zero_as_string: bool,
    pub xml_attr_prefix: String,
    pub xml_text_node_prop_name: String,
    pub empty_element_handling: NullValue,
}
```

Image captured from:

https://docs.rs/quickxml_to_serde/latest/quickxml_to_serde/struct.Config.html

For example, if `leading_zero_as_string` is `true`, all the numbers in the xml file with leading 0, e.g., 005, will be treated as a string instead of an integer so the 0 is not trimmed away. The `new_with_custom_values()` sets up the `Config` struct with:

- "`leading_zero_as_string: true`" so the leading 0 is not lost.
- "`xml_attr_prefix: ""`" so there is no extra prefix for the json attribute (i.e. `<x a="Hello!" />` becomes `{"x": {"a":"Hello!"}}` and not something like `{"x": {"@a":"Hello!"}}`. This is to make the output file meet the common user preference by not altering their data.
- "`xml_text_node_prop_name: "txt"`" to set up text node's property name. For example: `<x a="Hello!">Goodbye!</x>` to become `{"x": {"a":"Hello!", "txt":"Goodbye!"}}`
- "`empty_element_handling: NullValue::Null`". If there is an empty element like `<x />`, the program will replace it with the `Null` variant of `NullValue`. The output of `<x />` will be: `"x":null`.

Following this, json is obtained using `xml_string_to_json()` in the form of `Result<Value, Error>`.

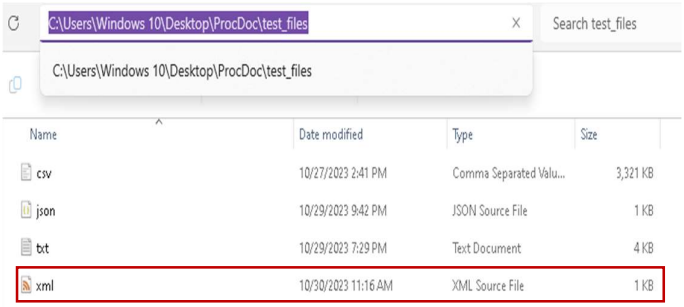
```rust
    let mut json_file: File = File::create(path: format!(
        "{}/{}.{}",
        info.output_path, info.name_file, info.output_extension
    ))?;
    json_file.write_all(buf: json.unwrap().to_string().as_bytes())?;

    Ok(())
```

The output file is created in the same way as other function, `Value` is `unwrap`ped and converted to `String`. Finally, the output file is written with the json content.

Input:

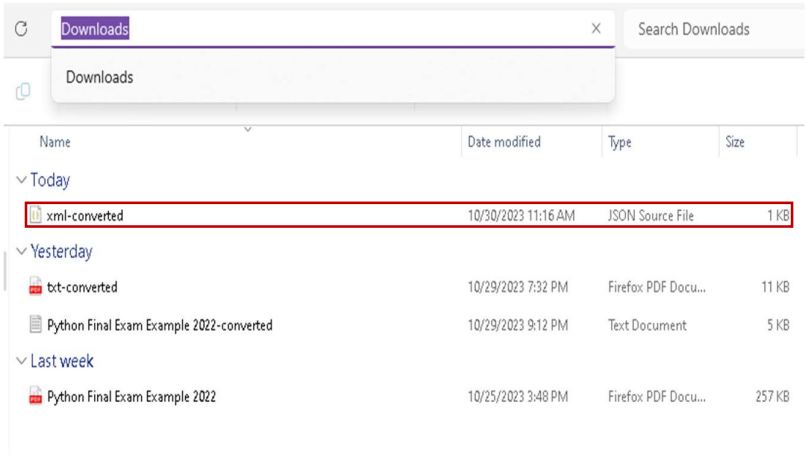
```xml
<?xml version="1.0" encoding="utf-8"?>
<quiz>
    <maths>
        <q1>
            <answer>12</answer>
            <options>10</options>
            <options>11</options>
            <options>12</options>
            <options>13</options>
            <question>5 + 7 = ?</question>
        </q1>
        <q2>
            <answer>4</answer>
            <options>1</options>
            <options>2</options>
            <options>3</options>
            <options>4</options>
            <question>12 - 8 = ?</question>
        </q2>
    </maths>
    <sport>
        <q1>
            <answer>Huston Rocket</answer>
            <options>New York Bulls</options>
            <options>Los Angeles Kings</options>
            <options>Golden State Warriros</options>
            <options>Huston Rocket</options>
            <question>Which one is correct team name in NBA?</question>
        </q1>
    </sport>
</quiz>
```

```
PS C:\Users\Windows 10\Desktop\ProcDoc> cargo run -- -i "test_files/xml.xml" -e json
   Compiling proc_doc v0.1.0 (C:\Users\Windows 10\Desktop\ProcDoc)
    Finished dev [unoptimized + debuginfo] target(s) in 2.35s
     Running `target\debug\proc_doc.exe -i test_files/xml.xml -e json`
Success!
```



Output:

```json
{
    "quiz": {
        "maths": {
            "q1": {
                "answer": {
                    "o": "Hello",
                    "txt": 12
                },
                "options": [
                    10,
                    11,
                    12,
                    13
                ],
                "question": "5 + 7 = ?"
            },
            "q2": {
                "answer": 4,
                "options": [
                    1,
                    2,
                    3,
                    4
                ],
                "question": "12 - 8 = ?"
            }
        },
        "sport": {
            "q1": {
                "answer": "Huston Rocket",
                "options": [
                    "New York Bulls",
                    "Los Angeles Kings",
                    "Golden State Warriros",
                    "Huston Rocket"
                ],
                "question": "Which one is correct team name in NBA?"
            }
        }
    }
}
```



Function 6: txt to docx

```
use docx_rs::*;
```

This function uses **`docx_rs`** crate v0.4.7.

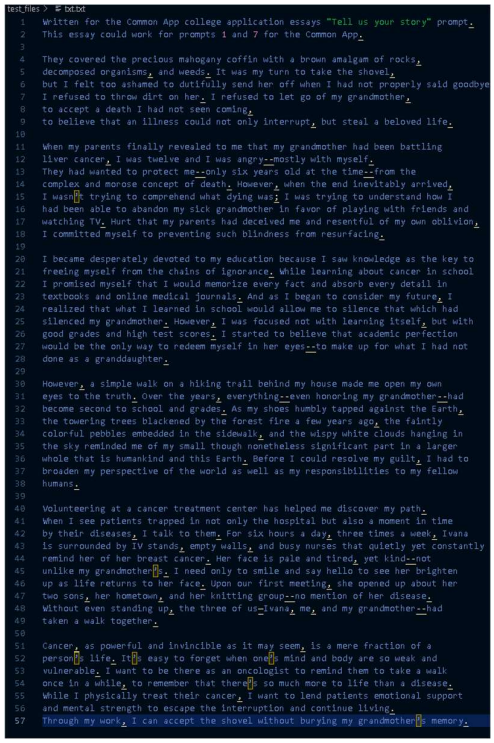
```
fn txt_to_docx(info: ConversionInfo) -> MyResultBox {
    let mut txt_content: String = String::new();
    File::open(&info.input_path)?.read_to_string(buf: &mut txt_content)?;

    let word_docx: Docx = Docx::new();
    let paragraph: Paragraph = Paragraph::new().add_run(Run::new().add_text(txt_content.clone()));
```

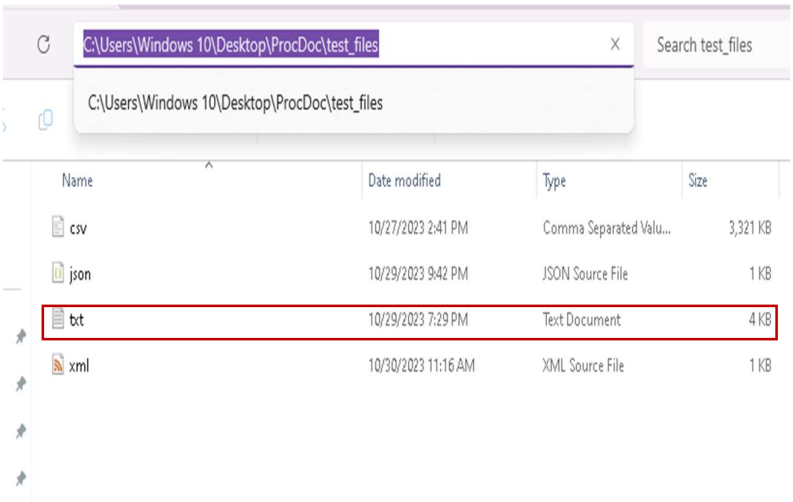
The txt file is read using **`std::fs`**. A variable **`word_docx`** is declared with **`Docx`** object and **`paragraph`** is assigned with a **`Paragraph`** object with **`Runs`** containing text content read from the txt file.

```
    let docx_file: File = File::create(path: format!(
        "{}/{}.{}",
        info.output_path, info.name_file, info.output_extension
    ))?;
    word_docx.add_paragraph(paragraph).build().pack(docx_file)?;

    Ok(())
```

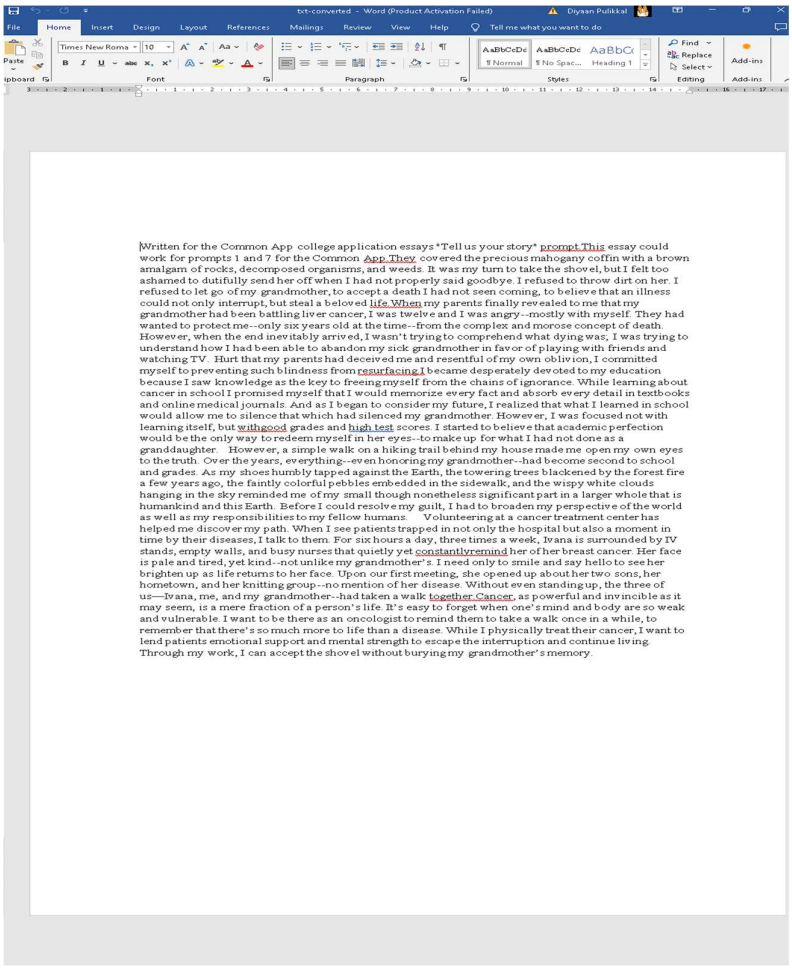
A new docx file is created and the **`Docx`** object **`word_docx`** has the **`paragraph`** added as its property. Finally, **`build()`** and **`pack()`** which convert it to an XMLDocx object and zip itself into the file respectively.

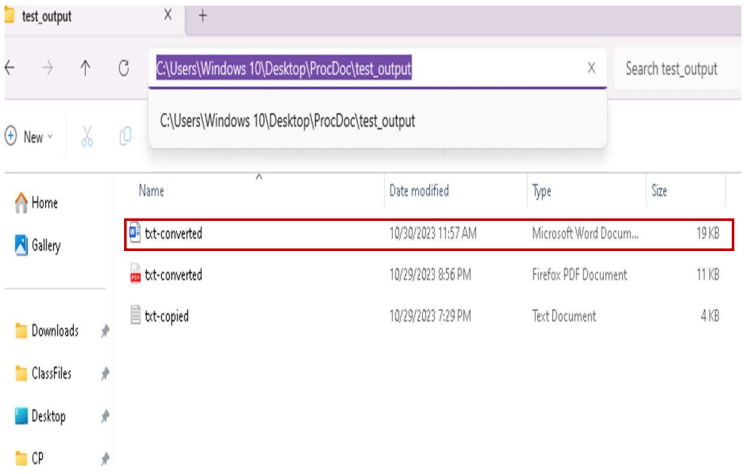
Input:





Output:


txt-converted - Word (Product Activation Failed)
Diyaan Pulikkal
File Home Insert Design Layout References Mailings Review View Help Tell me what you want to do
Written for the Common App college application essays "Tell us your story" prompt.This essay could work for prompts 1 and 7 for the Common App.They covered the precious mahogany coffin with a brown amalgam of rocks, decomposed organisms, and weeds. It was my turn to take the shovel, but I felt too ashamed to dutifully send her off when I had not properly said goodbye. I refused to throw dirt on her. I refused to let go of my grandmother, to accept a death I had not seen coming, to believe that an illness could not only interrupt, but steal a beloved life.When my parents finally revealed to me that my grandmother had been battling liver cancer, I was twelve and I was angry--mostly with myself. They had wanted to protect me--only six years old at the time--from the complex and morose concept of death. However, when the end inevitably arrived, I wasn't trying to comprehend what dying was, I was trying to understand how I had been able to abandon my sick grandmother in favor of playing with friends and watching TV. Hurt that my parents had deceived me and resentful of my own oblivion, I committed myself to preventing such blindness from resurfacing.I became desperately devoted to my education because I saw knowledge as the key to freeingmyself from the chains of ignorance. While learning about cancer in school I promised myself that I would memorize every fact and absorb every detail in textbooks and online medical journals. And as I began to consider my future, I realized that what I learned in school would allow me to silence that which had silenced my grandmother. However, I was focused not with learning itself, but withgood grades and high test scores. I started to believe that academic perfection would be the only way to redeem myself in her eyes--to make up for what I had not done as a granddaughter. However, a simple walk on a hiking trail behind my house made me open my own eyes to the truth. Over the years, everything--even honoring my grandmother--had become second to school and grades. As my shoes humbly tapped against the Earth, the towering trees blackened by the forest fire a few years ago, the faintly colorful pebbles embedded in the sidewalk, and the wispy white clouds hanging in the sky reminded me of my small though nonetheless significant part in a larger whole that is humankind and this Earth. Before I could resolve my guilt, I had to broaden my perspective of the world as well as my responsibilities to my fellow humans. Volunteering at a cancer treatment center has helped me discover my path. When I see patients trapped in not only the hospital but also a moment in time by their diseases, I talk to them. For six hours a day, three times a week, Ivana is surrounded by IV stands, empty walls, and busy nurses that quietly yet constantlyremind her of her breast cancer. Her face is pale and tired, yet kind--not unlike my grandmother's. I need only to smile and say hello to see her brighten up as life returns to her face. Upon our first meeting, she opened up about her two sons, her hometown, and her knitting group--no mention of her disease. Without even standing up, the three of us—Ivana, me, and my grandmother--had taken a walk together.Cancer, as powerful and invincible as it may seem, is a mere fraction of a person's life. It's easy to forget when one's mind and body are so weak and vulnerable. I want to be there as an oncologist to remind them to take a walk once in a while, to remember that there's so much more to life than a disease. While I physically treat their cancer, I want to lend patients emotional support and mental strength to escape the interruption and continue living. Through my work, I can accept the shovel without burying my grandmother's memory.


test_output
C:\Users\Windows 10\Desktop\ProcDoc\test_output
Search test_output
New
Home
Gallery
Downloads
ClassFiles
Desktop
CP
Name
Date modified
Type
Size
txt-converted
10/30/2023 11:57 AM
Microsoft Word Docum...
19 KB
txt-converted
10/29/2023 8:56 PM
Firefox PDF Document
11 KB
txt-copied
10/29/2023 7:29 PM
Text Document
4 KB

Function 7: csv to html

Explanation:

```
use csv;
```

This function uses the crate **`csv`** v1.3.0.

```
fn csv_to_html(info: ConversionInfo) -> MyResultBox {
    // Set up html filex
    let mut html_file: File = File::create(path: format!(
        "{}/{}.{}",
        info.output_path, info.name_file, info.output_extension
    ))?;
    html_file.write_all(buf: format!("<!DOCTYPE html><html><body><table><tr>").as_bytes())?;
```

Create an output html file to write the **`<!DOCTYPE…tr>`** structure into the html file.

```
    // Construct csv reader
    let mut csv_file: Reader<File> = csv::Reader::from_path(info.input_path)?;

    // Make column headers
    let headers: StringRecord = csv_file.headers()?.clone();
    let vec_header: Vec<String> = headers.iter().map(|s: &str| s.to_string()).collect();
    for i: String in vec_header {
        html_file.write_all(buf: format!("<th>{}</th>", i).as_bytes())?;
    }
    html_file.write_all(buf: "</tr>".as_bytes())?;
```

Next, the program creates a **`csv::Reader`** object that collects all the data from the input file. The program then iterates through the header of the csv file to construct columns with the associated header as column title.

```
  for i: Result<StringRecord, Error> in csv_file.records() {
      let vec_data: Vec<String> = i?.iter().map(|s: &str| s.to_string()).collect();
      html_file.write_all(buf: "<tr>".as_bytes())?;
      for j: String in vec_data {
          html_file.write_all(buf: format!("<td>{}</td>", j).as_bytes())?
      }
      html_file.write_all(buf: "</tr>".as_bytes())?;
  }
```

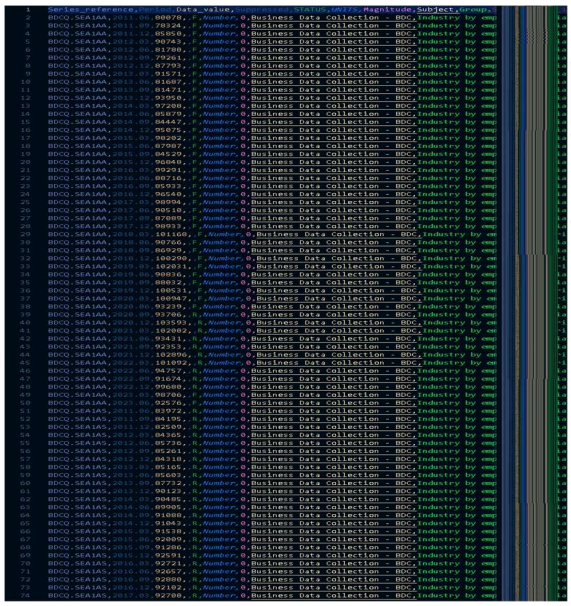
The parent **`for`** loop iterates through each row of the csv data and convert it to a **`Vec`** of **`String`** while also writing **`<tr>`** to declare the starting point of the table row. On the other hand, the child loop iterates through each element in the vector and write **`<td>element</td>`** into the html file. The program then closes the table row with **`</tr>`** tag.

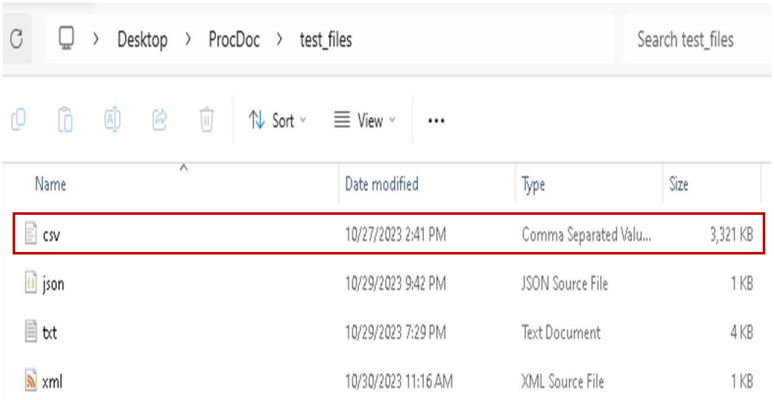
```
    // Finish up the html structure
    html_file.write_all(buf: format!("</table></body></html>").as_bytes())?;

    Ok(())
} fn csv_to_html
```

Finally, the program finishes up by closing the structure of **`<table>`**, **`<body>`** and **`<html>`** in order.

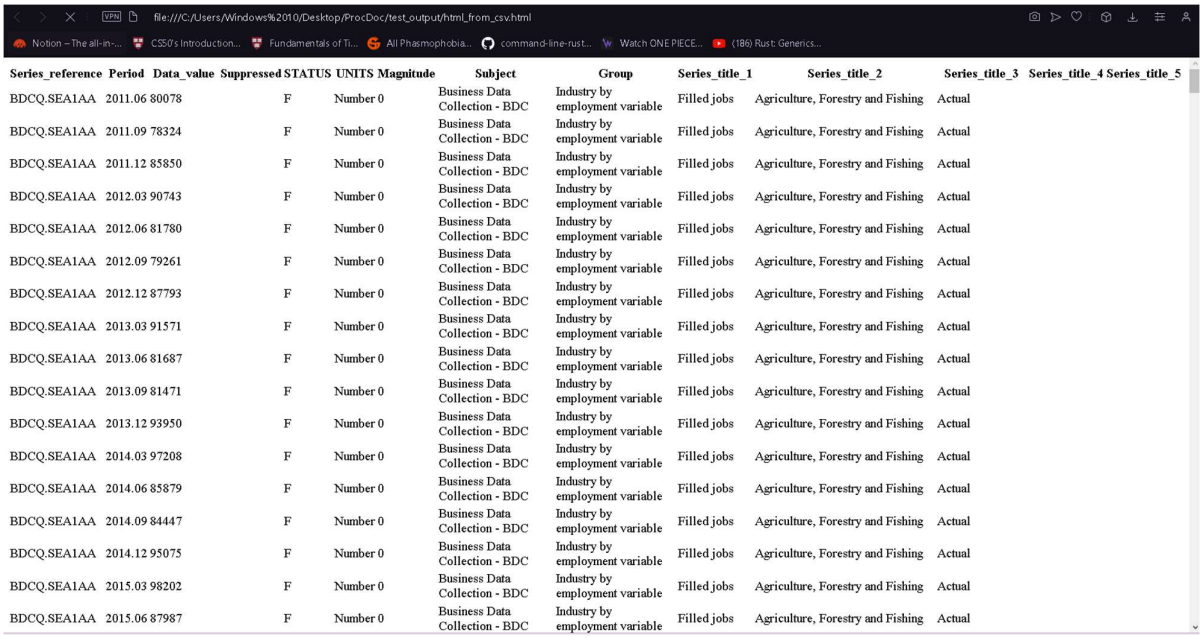
Input:

```
PS C:\Users\Windows 10\Desktop\ProcDoc> cargo run -- -i "test_files/csv.csv" -e html -o "test_output"
-n html_from_csv
   Compiling proc_doc v0.1.0 (C:\Users\Windows 10\Desktop\ProcDoc)
    Finished dev [unoptimized + debuginfo] target(s) in 2.56s
warning:
     Running `target\debug\proc_doc.exe -i test_files/csv.csv -e html -o test_output -n html_from_csv`
Success!
```





Output:



Series_reference	Period	Data_value	Suppressed	STATUS	UNITS	Magnitude	Subject	Group	Series_title_1	Series_title_2	Series_title_3	Series_title_4	Series_title_5
BDCQ.SEA1AA	2011.06	80078		F	Number	0	Business Data Collection - BDC	Industry by employment variable	Filled jobs	Agriculture, Forestry and Fishing	Actual		
BDCQ.SEA1AA	2011.09	78324		F	Number	0	Business Data Collection - BDC	Industry by employment variable	Filled jobs	Agriculture, Forestry and Fishing	Actual		
BDCQ.SEA1AA	2011.12	85850		F	Number	0	Business Data Collection - BDC	Industry by employment variable	Filled jobs	Agriculture, Forestry and Fishing	Actual		
BDCQ.SEA1AA	2012.03	90743		F	Number	0	Business Data Collection - BDC	Industry by employment variable	Filled jobs	Agriculture, Forestry and Fishing	Actual		
BDCQ.SEA1AA	2012.06	81780		F	Number	0	Business Data Collection - BDC	Industry by employment variable	Filled jobs	Agriculture, Forestry and Fishing	Actual		
BDCQ.SEA1AA	2012.09	79261		F	Number	0	Business Data Collection - BDC	Industry by employment variable	Filled jobs	Agriculture, Forestry and Fishing	Actual		
BDCQ.SEA1AA	2012.12	87793		F	Number	0	Business Data Collection - BDC	Industry by employment variable	Filled jobs	Agriculture, Forestry and Fishing	Actual		
BDCQ.SEA1AA	2013.03	91571		F	Number	0	Business Data Collection - BDC	Industry by employment variable	Filled jobs	Agriculture, Forestry and Fishing	Actual		
BDCQ.SEA1AA	2013.06	81687		F	Number	0	Business Data Collection - BDC	Industry by employment variable	Filled jobs	Agriculture, Forestry and Fishing	Actual		
BDCQ.SEA1AA	2013.09	81471		F	Number	0	Business Data Collection - BDC	Industry by employment variable	Filled jobs	Agriculture, Forestry and Fishing	Actual		
BDCQ.SEA1AA	2013.12	93950		F	Number	0	Business Data Collection - BDC	Industry by employment variable	Filled jobs	Agriculture, Forestry and Fishing	Actual		
BDCQ.SEA1AA	2014.03	97208		F	Number	0	Business Data Collection - BDC	Industry by employment variable	Filled jobs	Agriculture, Forestry and Fishing	Actual		
BDCQ.SEA1AA	2014.06	85879		F	Number	0	Business Data Collection - BDC	Industry by employment variable	Filled jobs	Agriculture, Forestry and Fishing	Actual		
BDCQ.SEA1AA	2014.09	84447		F	Number	0	Business Data Collection - BDC	Industry by employment variable	Filled jobs	Agriculture, Forestry and Fishing	Actual		
BDCQ.SEA1AA	2014.12	95075		F	Number	0	Business Data Collection - BDC	Industry by employment variable	Filled jobs	Agriculture, Forestry and Fishing	Actual		
BDCQ.SEA1AA	2015.03	98202		F	Number	0	Business Data Collection - BDC	Industry by employment variable	Filled jobs	Agriculture, Forestry and Fishing	Actual		
BDCQ.SEA1AA	2015.06	87987		F	Number	0	Business Data Collection - BDC	Industry by employment variable	Filled jobs	Agriculture, Forestry and Fishing	Actual		

End of documentation