1. Title and Basic Details

* + Project Title:

Parser For University Form Submissions

* + - Build a parser to validate and process forms submitted by students (e.g., admission, examination, or feedback forms).
    - Use both top-down and bottom-up parsing techniques for robust validation.

1. Abstract

This C program validates and processes a student feedback form by checking the correctness of user inputs for several fields: Student ID, Feedback Date, Rating, and Feedback Comment. The program ensures that the Student ID is exactly 6 alphanumeric characters, the Feedback Date is in the correct YYYY-MM-DD format, the Rating is an integer between 1 and 5, and the Feedback Comment contains at least 6 characters

1. Objectives

this program aims to enhance form submission accuracy and ensure that only valid data is processed, contributing to effective data collection in applications such as surveys, evaluations, and administrative systems.

1. Methodology

* Design: Define the structure and constraints for the form.
* Validate: Create functions to validate each form field.
* Process: Handle the form processing logic after validation.
* Error Handling: Provide informative messages to guide users in correcting invalid inputs.

1.program with python code:

o Build a parser to validate and process forms submitted by students (e.g., admission, examination, or feedback forms).

import re

class FormParser: def \_init\_(self, form\_type): self.form\_type = form\_type self.required\_fields = [] self.field\_validations = {}

def set\_required\_fields(self, fields): """Set required fields for the form""" self.required\_fields = fields

def set\_field\_validations(self, field, validation\_func): """Set validation function for a specific field""" self.field\_validations[field] = validation\_func

def validate\_form(self, form\_data):

"""Validate the form data based on required fields and field validations""" errors = []

# Check if required fields are present for field in self.required\_fields: if field not in form\_data or not form\_data[field]:

errors.append(f"'{field}' is required.")

# Check if fields meet specific validations for field, validation\_func in self.field\_validations.items(): if field in form\_data and not validation\_func(form\_data[field]):

errors.append(f"'{field}' is invalid.")

return errors

def process\_form(self, form\_data):

"""Process and handle form data after validation""" errors = self.validate\_form(form\_data) if errors:

return {"status": "error", "messages": errors}

# Form data is valid, you can process it here (e.g., store it in a database) return {"status": "success", "data": form\_data}

# Example validation functions

@staticmethod def validate\_email(email):

"""Check if the email format is correct""" email\_regex = r'^[a-zA-Z0-9\_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.]+$' return bool(re.match(email\_regex, email))

@staticmethod def validate\_phone(phone):

"""Check if the phone number is valid (e.g., a simple 10-digit check)""" phone\_regex = r'^\d{10}$' return bool(re.match(phone\_regex, phone)) @staticmethod def validate\_age(age):

"""Check if age is a positive integer above a certain threshold"""

try:

age = int(age) return age > 18 # Let's assume

students must be over 18 except ValueError:

return False

# Create a form parser for admission form admission\_parser

= FormParser(form\_type="Admission")

# Define required fields

admission\_parser.set\_required\_fields(['name', 'email', 'phone', 'age'])

# Set field-specific validations

admission\_parser.set\_field\_validations('email', admission\_parser.validate\_email)

admission\_parser.set\_field\_validations('phone', admission\_parser.validate\_phone) admission\_parser.set\_field\_validations('age', admission\_parser.validate\_age)

# Example form data submitted by a student form\_data

= {

"name": "John Doe",

"email": "john.doe@example.com",

"phone": "1234567890",

"age": "22"

}

# Process the form

result = admission\_parser.process\_form(form\_data)

if result['status'] == 'success':

print("Form processed successfully!")

print("Form Data:", result['data']) else:

print("Errors in form submission:")

for message in result['messages']:

print(message)

2. program with C code:

o Use both top-down and bottom-up parsing techniques for robust validation.

import re

class FormParser: def \_init\_(self, form\_type): self.form\_type = form\_type self.required\_fields = [] self.field\_validations = {}

def set\_required\_fields(self, fields): """Set required fields for the form""" self.required\_fields = fields

def set\_field\_validations(self, field, validation\_func): """Set validation function for a specific field""" self.field\_validations[field] = validation\_func

def top\_down\_validation(self, form\_data):

"""Perform top-down validation to ensure required fields exist and structure is correct"""

errors = []

# Check if required fields are present for field in self.required\_fields:

if field not in form\_data or not form\_data[field]: errors.append(f"'{field}' is required.")

return errors

def bottom\_up\_validation(self, form\_data):

"""Perform bottom-up validation to ensure field-specific rules are followed""" errors = []

# Check if fields meet specific validations for field, validation\_func in self.field\_va lidations.items(): if field in form\_data and not validation\_func(form\_data[field]):

errors.append(f"'{field}' is invalid.")

return errors

def validate\_form(self, form\_data):

"""Validate the form data using both top-down and bottom-up techniques""" top\_down\_errors = self.top\_down\_validation(form\_data) bottom\_up\_errors = self.bottom\_up\_validation(form\_data)

return top\_down\_errors + bottom\_up\_errors

def process\_form(self, form\_data):

"""Process and handle form data after validation""" errors = self.validate\_form(form\_data) if errors:

return {"status": "error", "messages": errors}

# Form data is valid, you can process it here (e.g., store it in a database) return {"status": "success", "data": form\_data}

# Example validation functions

@staticmethod def

validate\_email(email):

"""Check if the email format is correct""" email\_regex = r'^[a-zA-Z0-9\_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.]+$' return bool(re.match(email\_regex, email))

@staticmethod def validate\_phone(phone):

"""Check if the phone number is valid (e.g., a simple 1 0-digit check)""" phone\_regex = r'^\d{10}$' return bool(re.match(phone\_regex, phone))

@staticmethod def validate\_age(age):

"""Check if age is a positive integer above a certain threshold""" try:

age = int(age) return age > 18 # Let's assume

students must be over 18 except ValueError:

return False

# Example usage:

# Create a form parser for admission form admission\_parser = FormParser(form\_type="Admission") # Define required fields admission\_parser.set\_required\_fields(['name', 'email',

'phone', 'age'])

# Set field-specific validations

admission\_parser.set\_field\_validations('email', admission\_parser.validate\_email)

admission\_parser.set\_field\_validations('phone', admission\_parser.validate\_phone) admission\_parser.set\_field\_validations('age', admission\_parser.validate\_age)

# Example form data submitted by a student form\_data

= {

"name": "John Doe",

"email": "john.doe@example.com",

"phone": "1234567890",

"age": "22"

}

# Process the form

result = admission\_parser.process\_form(form\_data)

if result['status'] == 'success':

print("Form processed successfully!")

print("Form Data:", result['data']) else:

print("Errors in form submission:")

for message in result['messages']:

print(message)import re

class FormParser: def \_init\_(self, form\_type): self.form\_type = form\_type self.required\_fields = [] self.field\_validations = {}

def set\_required\_fields(self, fields): """Set required fields for the form""" self.required\_fields = fields

def set\_field\_validations(self, field, validation\_func): """Set validation function for a specific field""" self.field\_validations[field] = validation\_func

def top\_down\_validation(self, form\_data):

"""Perform top-down validation to ensure required fields exist and structure is correct"""

errors = []

# Check if required fields are present for field in self.required\_fields: if field not in form\_data or not form\_data[field] errors.append(f"'{field}' is required.")

return errors

def bottom\_up\_validation(self, form\_data):

"""Perform bottom-up validation to ensure field-specific rules are followed""" errors = []

# Check if fields meet specific validations for field, validation\_func in self.field\_validations.items(): if field in form\_data and not validation\_func(form\_data[field]):

errors.append(f"'{field}' is invalid.") return errors

def validate\_form(self, form\_data):

"""Validate the form data using both top-down and bottom-up techniques""" top\_down\_errors = self.top\_down\_validation(form\_data) bottom\_up\_errors = self.bottom\_up\_validation(form\_data)

return top\_down\_errors + bottom\_up\_errors

def process\_form(self, form\_data):

"""Process and handle form data after validation""" errors = self.validate\_form(form\_data) if errors:

return {"status": "error", "messages": errors}

# Form data is valid, you can process it here (e.g., store it in a database) return {"status": "success", "data": form\_data}

# Example validation functions

@staticmethod def validate\_email(email):

"""Check if the email format is correct""" email\_regex = r'^[a- zA-Z0-9\_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.]+$' return

bool(re.match(email\_regex, email))

@staticmethod def validate\_phone(phone):

"""Check if the phone number is valid (e.g., a simple 10-digit check)""" phone\_regex = r'^\d{10}$' return bool(re.match(phone\_regex, phone))

@staticmethod def validate\_age(age):

"""Check if age is a positive integer above a certain threshold""" try:

age = int(age) return age > 18 # Let's assume

students must be over 18 except ValueError:

return False

# Example usage:

# Create a form parser for admission form admission\_parser

= FormParser(form\_type="Admission")

# Define required fields

admission\_parser.set\_required\_fields(['name', 'email', 'phone', 'age'])

# Set field-specific validations

admission\_parser.set\_field\_validations('email', admission\_parser.validate\_email)

admission\_parser.set\_field\_validations('phone', admission\_parser.validate\_phone) admission\_parser.set\_field\_validations('age', admission\_parser.validate\_age)

# Example form data submitted by a student form\_data

= {

"name": "John Doe",

"email": "john.doe@example.com",

"phone": "1234567890",

"age": "22"

}

# Process the form

result = admission\_parser.process\_form(form\_data)

if result['status'] == 'success':

print("Form processed successfully!")

print("Form Data:", result['data']) else:

print("Errors in form submission:")

for message in result['messages']: print(message)import re

class FormParser: def \_init\_(self, form\_type): self.form\_type = form\_type self.required\_fields = [] self.field\_validations = {}

def set\_required\_fields(self, fields): """Set required fields for the form""" self.required\_fields = fields

def set\_field\_validations(self, field, validation\_func): """Set validation function for a specific field""" self.field\_validations[field] = validation\_func

def top\_down\_validation(self, form\_data):

"""Perform top-down validation to ensure requ ired fields exist and structure is correct""" errors = []

# Check if required fields are present for field in self.required\_fields: if field not in form\_data or not form\_data[field]:

errors.append(f"'{field}' is required.")

return errors

def bottom\_up\_validation(self, form\_data):

"""Perform bottom-up validation to ensure field-specific rules are followed""" errors = []

# Check if fields meet specific validations for field, validation\_func in self.field\_validations.items(): if field in form\_data and not validation\_func(form\_data[field]):

errors.append(f"'{field}' is invalid.")

return errors

def validate\_form(self, form\_data):

"""Validate the form data using both top-down and bottom-up techniques""" top\_down\_errors = self.top\_down\_validation(form\_data) bottom\_up\_errors = self.bottom\_up\_validation(form\_data)

return top\_down\_errors + bottom\_up\_errors

def process\_form(self, form\_data):

"""Process and handle form data after validation""" errors = self.validate\_form(form\_data)

if errors: return {"status": "error", "messages": errors}

# Form data is valid, you can process it here (e.g., store it in a database) return {"status": "success", "data": form\_data}

# Example validation functions

@staticmethod def

validate\_email(email):

"""Check if the email format is correct""" email\_regex = r'^[a-zA-Z0-9\_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.]+$' return bool(re.match(email\_regex, email))

@staticmethod def validate\_phone(phone):

"""Check if the phone number is valid (e.g., a simple 10-digit check)""" phone\_regex = r'^\d{10}$' return bool(re.match(phone\_regex, phone))

@staticmethod def validate\_age(age):

"""Check if age is a positive integer above a certain threshold""" try:

age = int(age) return age > 18 # Let's assume

students must be over 18 except ValueError:

return False

# Example usage:

# Create a form parser for admission form

admission\_parser = FormParser(form\_type="Admission")

# Define required fields

admission\_parser.set\_required\_fields(['name', 'email', 'phone', 'age'])

# Set field-specific validations

admission\_parser.set\_field\_validations('email', admission\_parser.validate\_email)

admission\_parser.set\_field\_validations('phone', admission\_parser.validate\_phone) admission\_parser.set\_field\_validations('age', admission\_parser.validate\_age)

# Example form data submitted by a student form\_data

= {

"name": "John Doe",

"email": "john.doe@example.com",

"phone": "1234567890",

"age": "22"

}

# Process the form result = admission\_parser.process\_form(form\_data)

if result['status'] == 'success':

print("Form processed successfully!")

print("Form Data:", result['data']) else:

print("Errors in form submission:")

for message in result['messages']:

print(message)import re

class FormParser: def \_init\_(self, form\_type): self.form\_type = form\_type self.required\_fields = [] self.field\_validations = {} def set\_required\_fields(self, fields): """Set required fields for the form""" self.required\_fields = fields

def set\_field\_validations(self, field, validation\_func ): """Set validation function for a specific field""" self.field\_validations[field] = validation\_func

def top\_down\_validation(self, form\_data):

"""Perform top-down validation to ensure required fields exist and structure is correct"""

errors = []

# Check if required fields are present for field in self.required\_fields:

if field not in form\_data or not form\_data[field]: errors.append(f"'{field}' is required.")

return errors

def bottom\_up\_validation(self, form\_data):

"""Perform bottom-up validation to ensure field-specific rules are followed""" errors = []

# Check if fields meet specific validations for field, validation\_func in self.field\_validations.items(): if field in form\_data and not validation\_func(form\_data[field]):

errors.append(f"'{field}' is invalid.")

return errors

def validate\_form(self, form\_data):

"""Validate the form data using both top-down and bottom-up techniques""" top\_down\_errors = self.top\_down\_validation(form\_data) bottom\_up\_errors = self.bottom\_up\_validation(form\_data)

return top\_down\_errors + bottom\_up\_errors def

process\_form(self, form\_data):

"""Process and handle form data after validation""" errors = self.validate\_form(form\_data) if errors:

return {"status": "error", "messages": errors}

# Form data is valid, you can process it here (e.g., store it in a database) return {"status": "success", "data": form\_data}

# Example validation functions

@staticmethod def validate\_email(email):

"""Check if the email format is correct""" email\_regex = r'^[a-zA-Z0-9\_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.]+$' return bool(re.match(email\_regex, email))

@staticmethod def validate\_phone(phone): """Check if the phone number is valid (e.g., a simple 10-digit check)""" phone\_regex = r'^\d{10}$'

return bool(re.match(phone\_regex, phone))

@staticmethod def validate\_age(age):

"""Check if age is a positive integer above a certain threshold""" try:

age = int(age) return age > 18 # Let's assume

students must be over 18 except ValueError:

return False

# Example usage:

# Create a form parser for admission form admission\_parser

= FormParser(form\_type="Admission")

# Define required fields admission\_parser.set\_required\_fields(['name', 'email', 'phone', 'age'])

# Set field-specific validations

admission\_parser.set\_field\_validations('email', admission\_parser.validate\_email)

admission\_parser.set\_field\_validations('phone', admission\_parser.validate\_phone) admission\_parser.set\_field\_validations('age', admission\_parser.validate\_age)

# Example form data submitted by a student form\_data

= {

"name": "John Doe",

"email": "john.doe@example.com",

"phone": "1234567890",

"age": "22"

}

# Process the form

result = admission\_parser.process\_form(form\_data)

if result['status'] == 'success':

print("Form processed successfully!")

print("Form Data:", result['data']) else:

print("Errors in form submission:")

for message in result['messages']: print(message)vimport re

class FormParser: def \_init\_(self, form\_type): self.form\_type = form\_type self.required\_fields = [] class FormParser: def \_init\_(self, form\_type): self.form\_type

= form\_type self.required\_fields = [] self.field\_validations = {}

def set\_required\_fields(self, fields): """Set required fields for the form""" self.required\_fields = fields

def set\_field\_validations(self, field, validation\_func): """Set validation function for a specific field""" self.field\_validations[field] = validation\_func

def top\_down\_validation(self, form\_data):

"""Perform top-down validation to ensure required fields exist and structure is correct"""

errors = []

# Check if required fields are present for field in self.required\_fields: if field not in form\_data or not form\_data[field]:

errors.append(f"'{field}' is required.")

return errors

def bottom\_up\_validation(self, form\_data):

"""Perform bottom-up validation to ensure field-specific rules are followed""" errors = []

# Check if fields meet specific validations for field, validation\_func in self.field\_validations.items():

if field in form\_data and not validation\_func(form \_data[field]): errors.append(f"'{field}' is invalid.")

return errors

def validate\_form(self, form\_data):

"""Validate the form data using both top-down and bottom-up techniques""" top\_down\_errors = self.top\_down\_validation(form\_data) bottom\_up\_errors = self.bottom\_up\_validation(form\_data)

return top\_down\_errors + bottom\_up\_errors

def process\_form(self, form\_data):

"""Process and handle form data after validation""" errors = self.validate\_form(form\_data) if errors:

return {"status": "error", "messages": errors}

# Form data is valid, you can process it here (e.g., store it in a database) return {"status": "success", "data": form\_data}

# Example validation functions

@staticmethod def validate\_email(email):

"""Check if the email format is correct""" email\_regex = r'^[a-zA-Z0-9\_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.]+$' return bool(re.match(email\_regex, email))

@staticmethod def validate\_phone(phone):

"""Check if the phone number is valid (e.g., a simple 10-digit check)""" phone\_regex = r'^\d{10}$' return bool(re.match(phone\_regex, phone))

@staticmethod

def validate\_age(age):

"""Check if age is a positive integer above a certain threshold""" try:

age = int(age) return age > 18 # Let's assume

students must be over 18 except ValueError:

return False

# Example usage:

# Create a form parser for admission form admission\_parser

= FormParser(form\_type="Admission")

# Define required fields

admission\_parser.set\_required\_fields(['name', 'email', 'phone', 'age'])

# Set field-specific validations admission\_parser.set\_field\_validations('email', admission\_parser.validate\_email) admission\_parser.set\_field\_validations('phone', admission\_parser.validate\_phone) admission\_parser.set\_field\_validations('age', admission\_parser.validate\_age)

# Example form data submitted by a student form\_data

= {

"name": "John Doe",

"email": "john.doe@example.com",

"phone": "1234567890",

"age": "22"

}

# Process the form

result = admission\_parser.process\_form(form\_data) if

result['status'] == 'success':

print("Form processed successfully!")

print("Form Data:", result['data']) else:

print("Errors in form submission:")

for message in result['messages']:

print(message)

* Result:
* Validate the Student ID as valid (6 alphanumeric characters).
* Validate the Feedback Date as valid (YYYY-MM-DD format).
* Validate the Rating as valid (5 is between 1 and 5).
* Validate the Feedback Comment as valid (it has more than 6 characters)

7. Conclusion

The two programs, one implemented in Python and the other in C, serve the same purpose of validating and processing a student feedback form. Both programs perform validation on the form fields, which include Student ID, Feedback Date, Rating, and Feedback Comment.