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.

2. 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

3. 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.

4. 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:

○ 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:

- 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.