## 1. Student Grade Calculator

Program Description: Create a program that calculates the final grade for a student based on assignments (30%), midterm exam (30%), and final exam (40%). The program should determine if the student passed (≥60%) or failed.

Key Features:

* Input assignment scores, midterm score, and final exam score
* Calculate weighted average based on predefined percentages
* Determine pass/fail status
* Display final grade and status to user

**Algorithm**

Step 1: start

Step 2: enter the assignment score

Step 3: enter the midterm exam score

Step 4: enter the final exam score

Step 5: calculate the final grade by

final grade=(assignment Score\*0.3)+(midterm score\*0.3)+(final exam score \*0.4)

Step 6: if the grade is greater than or equal to 60 print pass otherwise

fail

Step 7: print final grade

Step 8: stop

**pseudocode**

Begin

print”enter Assignment score(out of 100):”

read assignment score

print”enter midterm exam score(out of 100):”

read midterm score

print “enter final exam score(out of 100):”

read final exam score

final grade=(assignment score\*0.3)+(midterm score \*0.3)+(final exam score \*0.4)

print “final grade:”,final grade

if final grade >+ 60 then

print “pass”

else

print”fail”

endif

end

## 2. ATM Banking System

Program Description: Develop a program that simulates an ATM with options to check balance, deposit money, withdraw money, and exit. The program should maintain a running balance and prevent withdrawals that would result in a negative balance.

Key Features:

* Authenticate user with PIN
* Display menu of available operations
* Handle balance inquiries
* Process deposits and update balance
* Validate withdrawal requests against available balance
* Provide transaction receipts
* Allow user to exit system

**Algorithm**

Step 1: start

Step 2: set an initial account balance

Step 3: set a predefined pin for authentication

Step 4: give acces for users to enter their pin , if the entered pin is incorrect display the error message and allow retry, if the pin is correct ,display the menu options

* check balance

display the current balance

* deposite money

Ask for the deposit amount

add it to the balance

display the updated balance

* withdraw money

Ask for the withdrawal amount

if the amount is greater than the available balance,display insufficient funds and cancel the withdrawal

else subtract the amount from the balance and display the updated balance

Step 5: after completion return to the menu

Step 6: stop

**Pseudocode**

start

set balance =5000

set pin 1234

display”welcome to ATM”

input user pin

while user pin is not equal to pin:

display”incorrect pin.try again”

input user pin

do:

display”1.check balance”

display”2. Deposite money

display”3.withdraw money”

display “4.exit”

input choice

switch(choice):

case1:

display”your balance is :”,balance

case2:

display “enter deposite amount:”

input deposite

balance=balance + deposite

display”deposite successfull. New balance:”,balance

case 3:

display”enter withdrawal amount:”

input withdrawal

if withdrawal>balance then

display”insufficient balance”

else

balance = balance – withdrawal

display “withdrawal successful.new balance.”, balace

case 4:

display “thank you for using ATM. Existing..”

exit

default:

display” invalid option please try again.”

while choice !=4

stop

## 3. Inventory Management System

Program Description: Design a program that manages a store's inventory by allowing users to add new items, update quantities, remove items, and display the current inventory. Each item should have an ID, name, price, and quantity.

Key Features:

* Add new products to inventory with unique IDs
* Update existing product information
* Remove products from inventory
* Search for products by ID or name
* Display current inventory status
* Track low stock items
* Generate inventory reports

**Algorithm**

Step 1: Create an empty list or dictionary to store product details

Step 2: Show options for adding, updating, removing, searching, displaying, and generating reports.

Add Product:

* Prompt user for item details: ID, name, price, and quantity.
* Ensure the ID is unique before adding.
* Store product details in the inventory.

Update Product:

* Search for the product by ID.
* If found, update name, price, or quantity.

Remove Product:

* Search for the product by ID.
* If found, remove it from the inventory.

Search Product:

* Allow searching by ID or name.
* Display product details if found.

Step 3:Show all products with details (ID, name, price, quantity).

Step 4:Identify and display products below a certain quantity threshold.

Step 5:Provide summary details such as total products, total value of inventory, and low stock items.

Step 6: Keep running until the user chooses to exit.

**Pseudocode**

BEGIN

DEFINE inventory as an empty dictionary

FUNCTION display\_menu()

PRINT "1. Add Product"

PRINT "2. Update Product"

PRINT "3. Remove Product"

PRINT "4. Search Product"

PRINT "5. Display Inventory"

PRINT "6. Track Low Stock Items"

PRINT "7. Generate Inventory Report"

PRINT "8. Exit"

FUNCTION add\_product()

INPUT product\_id, name, price, quantity

IF product\_id already exists in inventory

PRINT "Product ID must be unique!"

ELSE

ADD product to inventory

FUNCTION update\_product()

INPUT product\_id

IF product\_id exists in inventory

INPUT new name, price, quantity

UPDATE product details

ELSE

PRINT "Product not found!"

FUNCTION remove\_product()

INPUT product\_id

IF product\_id exists in inventory

DELETE product from inventory

ELSE

PRINT "Product not found!"

FUNCTION search\_product()

INPUT search\_key (ID or name)

SEARCH inventory for matching product

IF found

PRINT product details

ELSE

PRINT "Product not found!"

FUNCTION display\_inventory()

FOR each product in inventory

PRINT product details

FUNCTION track\_low\_stock()

FOR each product in inventory

IF quantity < threshold

PRINT product details

FUNCTION generate\_report()

COMPUTE total value of inventory

PRINT total products, inventory value, low stock items

WHILE True

CALL display\_menu()

INPUT user\_choice

IF user\_choice == 1

CALL add\_product()

ELSE IF user\_choice == 2

CALL update\_product()

ELSE IF user\_choice == 3

CALL remove\_product()

ELSE IF user\_choice == 4

CALL search\_product()

ELSE IF user\_choice == 5

CALL display\_inventory()

ELSE IF user\_choice == 6

CALL track\_low\_stock()

ELSE IF user\_choice == 7

CALL generate\_report()

ELSE IF user\_choice == 8

EXIT program

ELSE

PRINT "Invalid Choice! Try Again."

END

## 4. Prime Number Checker

Program Description: Create a program that determines whether a given number is prime or not. A prime number is only divisible by 1 and itself with no other factors.

Key Features:

* Accept numerical input from user
* Verify if input is valid (positive integer)
* Use efficient algorithm to check for primality
* Display result with explanation
* Option to check additional numbers

**Algorithm**

Step 1: Start

Step 2:Prompt the user to enter a positive integer.

Step 3:Ensure the input is a valid positive integer.

Step 4:If the number is less than 2, it's not prime.

Step 5:If the number is 2 or 3, it's prime.

Step 6:If the number is even and greater than 2, it's not prime.

Step 7:Iterate from 3 to √N (square root of N) with step 2:

Step 8:If the number is divisible by any number in this range, it's not prime.

Step 9:Print whether the number is prime or not, along with an explanation.

Step 10:Allow repeated checks until the user chooses to exit.

Step 11: Stop

**Pseudocode**

BEGIN

FUNCTION is\_prime(n)

IF n < 2

RETURN False

IF n is 2 or 3

RETURN True

IF n is even

RETURN False

FOR i from 3 to sqrt(n) with step 2

IF n is divisible by i

RETURN False

RETURN True

WHILE True

PRINT "Enter a positive integer (or 0 to exit):"

INPUT number

IF number == 0

EXIT program

IF number is not a valid positive integer

PRINT "Invalid input! Enter a positive integer."

CONTINUE

CALL is\_prime(number)

IF is\_prime returns True

PRINT number "is a prime number."

ELSE

PRINT number "is not a prime number."

ASK user if they want to check another number

END

## 5. Temperature Conversion Tool

Program Description: Develop a program that converts temperatures between Celsius, Fahrenheit, and Kelvin. The user should be able to select the input and output temperature scales.

Key Features:

* Accept temperature value input
* Allow selection of source unit (C, F, or K)
* Allow selection of target unit (C, F, or K)
* Perform accurate conversion using correct formulas
* Display converted result with appropriate unit
* Option for multiple conversions

Algorithm

Step 1: Start

Step 2:Show options for temperature scales (Celsius, Fahrenheit, Kelvin).

Step 3:Get temperature value from the user.

* + Get the source unit (C, F, or K).
  + Get the target unit (C, F, or K).

Step 4:Ensure the user enters a valid temperature and unit.

Step 5:Use the correct conversion formula based on source and target units:

* + - Celsius to Fahrenheit: F=(C×9/5)+32
    - Celsius to Kelvin: K=C+273.15
    - Fahrenheit to Celsius: C=(F−32)×5/9
    - Fahrenheit to Kelvin: K=(F−32)×5/9+273.15
    - Kelvin to Celsius: C=K−273.15
    - Kelvin to Fahrenheit: F=(K−273.15)×9/5+32
  + If source and target units are the same, return the original value.

Step 6:Show the result with the appropriate unit.

Step 7:Allow the user to perform multiple conversions until they choose to exit.

Step 8: stop

**Pseudocode**

BEGIN

FUNCTION convert\_temperature(value, from\_unit, to\_unit)

IF from\_unit == to\_unit

RETURN value

SWITCH (from\_unit, to\_unit)

CASE ("C", "F"):

RETURN (value \* 9/5) + 32

CASE ("C", "K"):

RETURN value + 273.15

CASE ("F", "C"):

RETURN (value - 32) \* 5/9

CASE ("F", "K"):

RETURN (value - 32) \* 5/9 + 273.15

CASE ("K", "C"):

RETURN value - 273.15

CASE ("K", "F"):

RETURN (value - 273.15) \* 9/5 + 32

END SWITCH

WHILE True

PRINT "Enter temperature value (or type 'exit' to quit):"

INPUT value

IF value is 'exit'

EXIT program

PRINT "Enter source unit (C, F, K):"

INPUT from\_unit

PRINT "Enter target unit (C, F, K):"

INPUT to\_unit

IF from\_unit or to\_unit is invalid

PRINT "Invalid unit! Choose C, F, or K."

CONTINUE

CALL convert\_temperature(value, from\_unit, to\_unit)

PRINT "Converted temperature: " result to\_unit

ASK user if they want another conversion

END

## 6. Library Book Management System

Program Description: Design a program that manages a library's book collection, allowing librarians to add books, remove books, check out books to members, and return books. Track availability status for each book.

Key Features:

* Maintain database of books (title, author, ISBN, status)
* Maintain database of library members
* Process for adding new books to collection
* Process for removing obsolete books
* Book checkout procedure with due dates
* Book return procedure with potential late fees
* Search functionality by title, author, or ISBN
* Report generation for overdue books

Algorithm

Step 1: start

Step 2:Create a list or dictionary to store book details: title, author, ISBN, and availability status. Step 3:Create a list or dictionary for library members.

Step 4:Show options for adding, removing, checking out, returning, searching, and reporting overdue books.

Step 5:Accept book details (title, author, ISBN).

Step 6:Check if the book already exists; if not, add it to the collection.

Step 7:Accept ISBN as input,If the book exists, remove it from the library.

Step 8:Accept member ID and ISBN,Check if the book is available,Assign a due date (e.g., 14 days from checkout),Update the book’s status to “Checked Out.”

Step 9:Accept ISBN and member ID,Check if the book was checked out,If overdue, calculate the late fee,Update the book’s status to “Available.”

Step 10:Allow searching by title, author, or ISBN,Display matching book details.

Step 11:Identify and list overdue books along with the corresponding members,Keep running until the user chooses to exit.

Step 12:

Pseudocode

BEGIN

DEFINE books as an empty dictionary (ISBN -> {title, author, status, due\_date})

DEFINE members as an empty dictionary (ID -> {name, borrowed\_books})

FUNCTION display\_menu()

PRINT "1. Add Book"

PRINT "2. Remove Book"

PRINT "3. Checkout Book"

PRINT "4. Return Book"

PRINT "5. Search Book"

PRINT "6. Generate Overdue Report"

PRINT "7. Exit"

FUNCTION add\_book()

INPUT title, author, ISBN

IF ISBN exists in books

PRINT "Book already exists!"

ELSE

ADD book to books with status "Available"

PRINT "Book added successfully!"

FUNCTION remove\_book()

INPUT ISBN

IF ISBN exists in books

REMOVE book from books

PRINT "Book removed successfully!"

ELSE

PRINT "Book not found!"

FUNCTION checkout\_book()

INPUT member\_ID, ISBN

IF ISBN not in books OR book status is not "Available"

PRINT "Book is not available!"

ELSE

SET book status to "Checked Out"

SET due\_date to current\_date + 14 days

ADD book to member's borrowed\_books list

PRINT "Book checked out successfully!"

FUNCTION return\_book()

INPUT member\_ID, ISBN

IF ISBN not in books OR book status is "Available"

PRINT "Book was not checked out!"

ELSE

IF current\_date > due\_date

CALCULATE late fee

PRINT "Late fee: $" late\_fee

SET book status to "Available"

REMOVE book from member's borrowed\_books list

PRINT "Book returned successfully!"

FUNCTION search\_book()

INPUT search\_key

SEARCH books by title, author, or ISBN

PRINT matching books

FUNCTION generate\_overdue\_report()

FOR each book in books

IF book is checked out AND current\_date > due\_date

PRINT book details and member information

WHILE True

CALL display\_menu()

INPUT user\_choice

IF user\_choice == 1

CALL add\_book()

ELSE IF user\_choice == 2

CALL remove\_book()

ELSE IF user\_choice == 3

CALL checkout\_book()

ELSE IF user\_choice == 4

CALL return\_book()

ELSE IF user\_choice == 5

CALL search\_book()

ELSE IF user\_choice == 6

CALL generate\_overdue\_report()

ELSE IF user\_choice == 7

EXIT program

ELSE

PRINT "Invalid Choice! Try Again."

END

## 7. Fibonacci Sequence Generator

Program Description: Create a program that generates the Fibonacci sequence up to a specified number of terms. The Fibonacci sequence starts with 0 and 1, and each subsequent number is the sum of the two preceding numbers.

Key Features:

* Accept number of terms to generate
* Validate input is reasonable (positive integer within limits)
* Generate sequence using efficient algorithm
* Display sequence with appropriate formatting
* Option to save sequence to file

**Algorithm**

Step 1:start

Step 2:Prompt the user to enter the number of terms.

Step 3:Validate that the input is a positive integer greater than 0.

Step 4:Initialize the first two terms: F(0) = 0, F(1) = 1.,Use an iterative approach to compute the sequence efficiently.

Step 5:For each term F(n) = F(n-1) + F(n-2) until the desired number of terms is reached.

Step 6:Print the sequence in a formatted manner.

Step 7:Ask the user if they want to save the sequence to a file,If yes, write the sequence to a text file.

Step 8:Prompt the user if they want to generate another sequence,Continue until the user chooses to exit

Step 9: stop

**Pseudocode**

BEGIN

FUNCTION generate\_fibonacci(n)

IF n == 1

RETURN [0]

ELSE IF n == 2

RETURN [0, 1]

ELSE

SET fibonacci\_list = [0, 1]

FOR i from 2 to n-1

APPEND fibonacci\_list[i-1] + fibonacci\_list[i-2] to fibonacci\_list

RETURN fibonacci\_list

WHILE True

PRINT "Enter the number of terms (positive integer, or 0 to exit):"

INPUT num\_terms

IF num\_terms == 0

EXIT program

IF num\_terms is not a valid positive integer

PRINT "Invalid input! Please enter a positive integer."

CONTINUE

CALL generate\_fibonacci(num\_terms) -> fibonacci\_sequence

PRINT "Fibonacci Sequence:", fibonacci\_sequence

PRINT "Do you want to save the sequence to a file? (yes/no)"

INPUT user\_choice

IF user\_choice is "yes"

OPEN "fibonacci\_sequence.txt" for writing

WRITE fibonacci\_sequence to file

PRINT "Sequence saved to fibonacci\_sequence.txt"

PRINT "Do you want to generate another sequence? (yes/no)"

INPUT another\_try

IF another\_try is "no"

EXIT program

END

## 8. Calendar Event Scheduler

Program Description: Develop a program that allows users to schedule events on a calendar. Users should be able to add events with dates, times, and descriptions, view all events, and delete events.

Key Features:

* Add events with title, date, time, and description
* Validate date and time inputs
* Store events in organized data structure
* Display events for a specific day, week, or month
* Search events by title or description
* Delete or modify existing events
* Set reminders for upcoming events
* Check for schedule conflicts

**Algorithm**

Step 1: start

Step 2:Create a dictionary or list to store events with fields:

* + Title, Date, Time, Description

Step 3:Ensure events are sorted by date and time for efficient retrieval.

Step 4:Add a new event ,View events (day/week/month) ,Search events ,Modify or delete an event ,Set reminders ,Check for schedule conflicts

Step 5:Accept title, date, time, and description as input,Validate date and time format.

Step 6: Check if there is an existing event at the same date & time (schedule conflict).If no conflict, store event in the database.

Step 7:Allow user to view events for:

* + A specific day
  + A specific week
  + A specific month

Step 8:Retrieve and display events in a structured format.

Step 9:Allow searching by title or description,Return all matching events.

Step 10:Accept title or date-time to identify the event. If modifying, allow editing of title, date, time, or description. If deleting, remove the event from storage.

Step 11:Prompt user for events they want reminders for. Store reminder settings (e.g., notify 1 hour before event).

Step 12:Before adding a new event, check if an event already exists at the same date & time. Notify the user of a conflict and prompt for resolution (reschedule or overwrite).

Step 13:Continue execution until the user chooses to exit.

Step 14:stop

**Pseudocode**

BEGIN

DEFINE events as an empty list (stores event dictionaries)

FUNCTION display\_menu()

PRINT "1. Add Event"

PRINT "2. View Events"

PRINT "3. Search Events"

PRINT "4. Modify Event"

PRINT "5. Delete Event"

PRINT "6. Set Reminder"

PRINT "7. Exit"

FUNCTION add\_event()

INPUT title, date, time, description

IF date or time is invalid

PRINT "Invalid date/time format! Try again."

RETURN

IF event already exists at date and time

PRINT "Schedule conflict! Choose a different time."

RETURN

ADD event to events list

PRINT "Event added successfully!"

FUNCTION view\_events()

INPUT filter\_type (day/week/month)

IF filter\_type is "day"

INPUT specific\_date

DISPLAY events on specific\_date

ELSE IF filter\_type is "week"

INPUT start\_date

DISPLAY events in that week

ELSE IF filter\_type is "month"

INPUT month, year

DISPLAY events in that month

FUNCTION search\_event()

INPUT search\_key

SEARCH events list by title or description

DISPLAY matching events

FUNCTION modify\_event()

INPUT event\_title or date-time

IF event exists

PRINT "Enter new details (leave blank to keep existing values)"

UPDATE event details

PRINT "Event updated successfully!"

ELSE

PRINT "Event not found!"

FUNCTION delete\_event()

INPUT event\_title or date-time

IF event exists

REMOVE event from events list

PRINT "Event deleted successfully!"

ELSE

PRINT "Event not found!"

FUNCTION set\_reminder()

INPUT event\_title or date-time

IF event exists

INPUT reminder\_time (e.g., 1 hour before)

STORE reminder

PRINT "Reminder set successfully!"

ELSE

PRINT "Event not found!"

WHILE True

CALL display\_menu()

INPUT user\_choice

IF user\_choice == 1

CALL add\_event()

ELSE IF user\_choice == 2

CALL view\_events()

ELSE IF user\_choice == 3

CALL search\_event()

ELSE IF user\_choice == 4

CALL modify\_event()

ELSE IF user\_choice == 5

CALL delete\_event()

ELSE IF user\_choice == 6

CALL set\_reminder()

ELSE IF user\_choice == 7

EXIT program

ELSE

PRINT "Invalid choice! Try again."

END