a) Discuss the following terms in programming; (6 marks)

i. Arrays

- Arrays are data structures that store a collection of elements, such as variables or values, under a single name.
- Elements in an array are accessed using an index, which starts at 0 for the first element.
- Arrays are useful for organizing and manipulating a set of related data.

ii. Initialisation

- Initialization refers to the process of assigning an initial value to a variable when it is declared.
- It is essential for preventing the use of undefined or garbage values and ensures that variables start with a known state.

iii. Debugging

- Debugging is the process of identifying and fixing errors or bugs in a program.
- Debugging tools, such as breakpoints, print statements, and debugging environments, help programmers locate and correct issues in their code.

iv. Local variable

- A local variable is a variable declared within a specific block of code, such as a function or a compound statement.
- It is only accessible within the scope where it is declared, providing encapsulation and preventing unintended interference with other parts of the program.

b) Preprocessor statements and the main method are key in a program. Explain what these are how important they are in a program. (7marks)

- **Preprocessor Statements:**
-
  - Preprocessor statements in languages like C and C++ are directives that are processed before the compilation of the code.
  - They are used for tasks such as including header files, defining constants, and conditional compilation.

- Examples include *#include*, *#define*, and *#ifdef*.

- **Main Method:**
  - The main method is a special function in many programming languages, such as Java and C++, that serves as the entry point of a program.
  - Execution of the program begins with the main method.
  - It typically contains the code to be executed when the program starts.
- **Importance:**
  - Preprocessor statements enable code modularity, reuse, and configuration.
  - The main method is crucial as it orchestrates the execution flow of the program, calling other functions and coordinating overall functionality.

c) Use examples of your choice to explain how the following operators are used in programming.

c) i. Logical operators

Logical operators (AND, OR, NOT) are used to perform logical operations between two or more Boolean values.

```
# Example in Python
a = True
b = False
print(a and b)  # Output: False
```

d) ii. Decrement operators

Decrement operators (e.g., `--`) decrease the value of a variable by 1.

```
// Example in C
int x = 5;
x--;
printf("%d", x);  // Output: 4
```

e) iii. Assignment operators

Assignment operators (e.g., `=`, `+=`, `-=`) are used to assign values to variables.

```
#include <stdio.h>
int main() {
    int a = 5, b = 10;
    // Basic assignment
    int result1 = a + b;
    printf("Result1: %d\n", result1);
    // Compound assignment
    a += b;
    printf("Result2: %d\n", a);
    return 0;
}
```

f) iv. Modulus operators

Modulus operator (`%`) returns the remainder of the division of two numbers.

```
#include <stdio.h>
int main() {
    int a = 17, b = 5;
    int result = a % b;
    printf("Result: %d\n", result);
    return 0;
}
```

A) What is an array?

An array is a collection of elements, where each element is identified by an index or a key.

B) Initialize a character array holding three values.

```
#include <stdio.h>

int main() {
    char charArray[3] = {'A', 'B', 'C'};

    // Access and print the values in the array
    for (int i = 0; i < 3; i++) {
        printf("charArray[%d] = %c\n", i, charArray[i]);
    }

    return 0;
}
```

C) Write a program of your choice to show the application of switch statements.

```
#include <stdio.h>

int main() {
    int dayNumber;

    printf("Enter a number (1-7) to represent the day of the week: ");
    scanf("%d", &dayNumber);

    switch (dayNumber) {
        case 1:
            printf("Sunday\n");
            break;
        case 2:
            printf("Monday\n");
            break;
        case 3:
            printf("Tuesday\n");
            break;
        case 4:
            printf("Wednesday\n");
            break;
        case 5:
            printf("Thursday\n");
            break;
        case 6:
            printf("Friday\n");
            break;
        case 7:
            printf("Saturday\n");
            break;
        default:
            printf("Invalid input. Please enter a number between 1 and 7.\n");
    }

    return 0;
}
```

D) At standard super shop, the information in the table below was captured for the different transactions on one of the days. Use the concept of two dimensional and one dimensional arrays to write

a program that stores this information in arrays and computes the total cost before outputing the full information.

ITEM	QUANTITY	UNIT	COST	TOTAL COST
Posho	6		2000	
Beans	4		3000	

```
#include <stdio.h>

int main() {
    // Define constants for array size
    const int NUM_ITEMS = 2;
    const int NUM_ATTRIBUTES = 4;

    // Define and initialize the array to store transaction information
    int transactions[NUM_ITEMS][NUM_ATTRIBUTES] = {
        {6, 2000, 0, 0},  // Quantity, Unit Cost, Total Cost, Unused
        {4, 3000, 0, 0}   // Quantity, Unit Cost, Total Cost, Unused
    };

    // Compute total cost and update the array
    for (int i = 0; i < NUM_ITEMS; i++) {
        transactions[i][2] = transactions[i][0] * transactions[i][1]; // Total Cost
    }

    // Output the information
    printf("ITEM\tQUANTITY\tUNIT COST\tTOTAL COST\n");
    for (int i = 0; i < NUM_ITEMS; i++) {
        printf("Item %d\t%d\t\t%d\t\t%d\n", i + 1, transactions[i][0], transactions[i][1],
transactions[i][2]);
    }

    return 0;
}
```

A) While programming we can have global and local variables. Describe each of these, backing up your explanation with an example.

Global Variables: These are variables declared outside of any function, and they can be accessed and modified by any part of the program. Global variables have a global scope, meaning they are visible to all functions in the program.

```
#include <stdio.h>


// Global variable

int globalVar = 10;


void printGlobal() {

    printf("Global variable: %d\n", globalVar);

}


int main() {

    printGlobal();  // Accessing the global variable

    return 0;

}
```

Local Variables: These are variables declared within a specific function or block, and they have a local scope. They are only accessible within the block where they are defined.

```
#include <stdio.h>


void printLocal() {

    // Local variable
```

```
    int localVar = 5;

    printf("Local variable: %d\n", localVar);

}



int main() {

    printLocal();  // Accessing the local variable

    // printf("%d\n", localVar); // Error: localVar not accessible here

    return 0;

}
```

B) State the activities involved in the programming process.

- **1. Problem Definition:** Clearly define the problem that the program is intended to solve.
- **2. Planning:** Develop a plan or algorithm to solve the problem.
- **3. Coding:** Write the program code based on the plan using a programming language.
- **4. Compilation:** Translate the source code into machine code or an intermediate code.
- **5. Testing:** Execute the program with various inputs to ensure it behaves as expected.
- **6. Debugging:** Identify and fix errors or bugs in the code.
- **7. Documentation:** Create documentation to explain the program's functionality, usage, and code structure.
- **8. Maintenance:** Make updates or modifications to the program as needed.

C) A company needs a C program to help them with selecting applicants for an interview of an IT programmer job they advertised in the newspaper.

They intend to select candidate who is between the age of 25 and 40. They want to only consider Ugandans for this job. The considered candidates must have scored more than 70% in programming. The program required should adress the problem in the following style. • Requests for entry of nationality from user. • If entry is not Uganadan, the system displays a message saying, "Sorry,only Ugandans are considered for this job!". • If the entry is Ugandan, the system requests for entry of year of birth of the applicant. • The system then computes the current age of the applicant. • It then checks the age. If the age happens to be outside the range 25-40, the system displays an error message saying; "Sorry, applicant is outside the required age bracket". • If the age is in the required range, the system requests for entry of the programming score. • If the score is below 70%, the system displays an error message saying; "Sorry, applicant is not skilled enough". • If the score is 70% and above, the system displays a message saying, "applicant has been selected for the interview".

i. Use a FLOW CHART to design a program that will be used to develop the program above.
ii. Use C programming language to write a program based on the above design

```
#include <stdio.h>

int main() {

    // Step 1: Request nationality from the user

    char nationality[20];

    printf("Enter nationality: ");

    scanf("%s", nationality);
```

```
// Step 2: Check if the entry is Ugandan
if (strcmp(nationality, "Ugandan") != 0) {
    printf("Sorry, only Ugandans are considered for this job!\n");
} else {
    // Step 3: Request year of birth
    int birthYear;
    printf("Enter year of birth: ");
    scanf("%d", &birthYear);

    // Step 4: Compute current age
    int currentYear = 2023; // Assuming the current year is 2023
    int age = currentYear - birthYear;

    // Step 5: Check age range
    if (age < 25 || age > 40) {
        printf("Sorry, applicant is outside the required age bracket.\n");
    } else {
        // Step 6: Request programming score
        float programmingScore;
        printf("Enter programming score: ");
        scanf("%f", &programmingScore);
 // Step 7: Check programming score
        if (programmingScore < 70) {
            printf("Sorry, applicant is not skilled enough.\n");
```

```
            } else {
                // Step 8: Display selection message
                printf("Applicant has been selected for the interview.\n");
            }
        }
    }

    return 0;
}
```

Explain the concept of comments in C programming. Show how they are applied.

a) **Comments in C Programming:** Comments in C programming are used to provide additional information within the source code. They are not executed by the compiler and are intended for the human reader. Comments are helpful for explaining the code, making it more readable, and providing documentation. In C, there are two types of comments:

- **Single-line comments:** These start with `//` and continue until the end of the line.
-

```
// This is a single-line comment

int x = 5; // This is also a single-line comment
```

Multi-line comments: These are enclosed between `/*` and `*/` and can span multiple lines.

```
/*
 * This is a multi-line comment
```

```
 * It can span multiple lines
 */
```

b) How diifferent is the syntax error from a logical error. How can each be solved.

- **Syntax Error:** These are errors in the structure of the code. The compiler catches them during the compilation phase. They often involve incorrect use of keywords, missing or misplaced punctuation, etc. To fix syntax errors, carefully review the code and correct any mistakes in the syntax.
-
- **Logical Error:** These errors do not result in immediate failures, but the program does not produce the expected output. They are bugs in the algorithm or the overall logic of the code. Debugging tools and logical reasoning are used to identify and fix logical errors.
-

c) Write a program which outputs even numbers starting from 1000 to 100.

Program to Output Even Numbers:

```
#include <stdio.h>


int main() {
    int i;


    // Output even numbers from 1000 to 100
    for (i = 1000; i >= 100; i -= 2) {
        printf("%d\n", i);
```

```
    }

    return 0;
}
```

d) Uganda Christian University intends to increase its intake by 10% every academic year. Assuming the current number of students is 2000. Use a loop structure of your choice to show what the student number will be per year for the next 10 years.

d) **Program for Student Intake Projection:**

```
#include <stdio.h>

int main() {
    int currentStudents = 2000;
    int years = 10;
    float increasePercentage = 0.10;

    printf("Year\tStudent Count\n");

    for (int year = 1; year <= years; ++year) {
        printf("%d\t%d\n", year, currentStudents);
        currentStudents += (int)(currentStudents * increasePercentage);
```

```
    }


    return 0;

}
```

Discuss the following terms in programming;

i. Variables declaration

In programming, variable declaration is the process of specifying the data type and name of a variable that will be used to store values in a program. It allocates memory space for the variable and defines the type of data it can hold. For example, in C programming, you might declare a variable like this:

ii. Debugging

Debugging is the process of finding and fixing errors or bugs in a program. It involves identifying the cause of unexpected behavior, runtime errors, or logical issues in the code. Programmers use debugging tools, print statements, and other techniques to locate and correct errors, ensuring that the program behaves as intended.

iii. Constant

A constant is a value that doesn't change during the execution of a program. Constants are used to represent fixed values that are not meant to be modified. In C programming, you might declare a constant using the `const` keyword:`const int MAX_SIZE = 100;`

b) Explain the different parts of a program. Write a basic c program structure to back up your explanation.

```
// Preprocessor Directives

#include <stdio.h>

// Function Declaration (if any)
```

```
// main function (entry point of the program)

int main() {

    // Variable Declaration



    // Statements and Expressions



    // Return statement

    return 0;

}
```

- **Preprocessor Directives:** These are commands to the preprocessor, which processes the source code before actual compilation. `#include` is a common directive to include standard libraries.
- **Function Declaration:** If you define functions other than `main`, you need to declare them before using them.
- **Main Function:** This is the entry point of the program. Execution starts from the `main` function

c) Using your knowledge of operators. Write the output of each of these statements and explain your answer. For each number, assume the values for each of these variables to be as follows; int x = 7, g = 3, h = 17; i. w = x = g = h; ii. w = (++x); iii. w = ++h % g; iv.w

***= x; v. w = x % h * g + 8; vi. w = ((x < = h) | | (g = = h));**

Assuming the given values: `int x = 7, g = 3, h = 17;`

i. w = x = g = h;

- Explanation: This is an assignment from right to left. `h` gets assigned to `g`, then `g` to `x`, and finally, `x` to `w`.
- Output: `w`, `x`, and `g` will all be `17`.

ii. w = (++x);

- Explanation: Pre-increment `++x` increments the value of `x` before assignment.
- Output: `w` will be `8` (after incrementing `x`).

iii. w = ++h % g;

- Explanation: Pre-increment `++h` increments `h` before the modulus operation, then the result is assigned to `w`.
- Output: `w` will be `2` (after incrementing `h` and taking the modulus with `g`).

*iv. w = x;

- Explanation: Compound assignment `*=`, equivalent to `w = w * x`.
- Output: `w` will be `8` (previous value of `w` multiplied by `x`).

v. w = x % h * g + 8;

- Explanation: Evaluation follows the operator precedence. Modulus is performed first, then multiplication, addition, and assignment.
- Output: `w` will be `21`.

v. w = ((x <= h) || (g == h));

- Explanation: Logical OR (||) checks if either condition is true.
- Output: `w` will be `1` (true) because the second condition (`g == h`) is false, but the first (`x <= h`) is true. `1` represents true in C.

Flow charts are the most used design models used in structured programming. Use a program of your choice to show the use of flow charts.

b) Differentiate the compilation stage from the deburging stage in programming.

.

Compilation Stage:

.

- **Purpose:** The compilation stage is the process of translating human-readable source code into machine-readable code or bytecode. It is a crucial step in the software development lifecycle.
- **Activities:** During compilation, the compiler analyzes the entire source code, checks for syntax errors, and translates the code into an intermediate form or directly into machine code, depending on the programming language.
- **Output:** The output of the compilation stage is typically an executable file, bytecode, or another form of intermediate code that can be executed by the computer's hardware.

.

Debugging Stage:

.

- **Purpose:** Debugging is the process of identifying, analyzing, and fixing errors (bugs) or unexpected behavior in the software. The goal is to ensure that the program functions correctly and produces the expected results.
- **Activities:** During debugging, developers use various tools and techniques to locate and understand the source of errors. This involves inspecting variables, stepping through code, setting breakpoints, and employing debugging tools to identify and correct issues.
- **Output:** The output of the debugging stage is a refined and error-free version of the source code. Debugging may involve making

changes to the code, fixing logical errors, or addressing issues related to program flow.

c) Recently there was a scuffle as some youths were voting concerning the amendment of the bill concerning the lifting of the age limit for presidency in Uganda. A software was used to ease the process. Each youth was required to enter a number to show if they want the bill ammended or not. Whenever a number for YES was entered, the system displayed "GIKWATEKO.....!!!" three times and when NO was entered, the system displayed "TOGOKWATAKO.....!!!" three times. (1 for YES, 2 for NO). This excited the youth and encouraged many to use the system and cast their vote in regard to this bill amendment.

i. Write a c program which you think was used to achieve this system behaviour using the IF structure.

Using *if* Structure:

```
#include <stdio.h>

int main() {

    int vote;

  // Get the vote input from the user

    printf("Enter your vote (1 for YES, 2 for NO): ");

    scanf("%d", &vote);



    // Check the vote and display the corresponding message

    if (vote == 1) {
```

```
        printf("GIKWATEKO.....!!!\n");
        printf("GIKWATEKO.....!!!\n");
        printf("GIKWATEKO.....!!!\n");
    } else if (vote == 2) {
        printf("TOGOKWATAKO.....!!!\n");
        printf("TOGOKWATAKO.....!!!\n");
        printf("TOGOKWATAKO.....!!!\n");
    } else {
        printf("Invalid vote! Please enter 1 for YES or 2 for NO.\n");
    }

    return 0;
}
```

iii. Re-write the above program using a switch structure .

```
#include <stdio.h>

int main() {
    int vote;

    // Get the vote input from the user
    printf("Enter your vote (1 for YES, 2 for NO): ");
    scanf("%d", &vote);
```

```
    // Use switch to check the vote and display the corresponding message
    switch (vote) {
        case 1:
            printf("GIKWATEKO.....!!!\n");
            printf("GIKWATEKO.....!!!\n");
            printf("GIKWATEKO.....!!!\n");
            break;
        case 2:
            printf("TOGOKWATAKO.....!!!\n");
            printf("TOGOKWATAKO.....!!!\n");
            printf("TOGOKWATAKO.....!!!\n");
            break;
        default:
            printf("Invalid vote! Please enter 1 for YES or 2 for NO.\n");
    }

    return 0;
}
```

Initialise a character array holding a name of your choice. b) int results[4] = { 50 , 68 , 90 }; Basing on the statement above, compute the average of the values shown in the given memory loacation. Store this average in the last position of the memory location. c) KIKO market has different stall owners who sell different types of items like fruits, meat,

food. Assume the different stalls pay the following daily fees to municiple council.➢ Fruit stalls pay UGX. 1000➢ Food stalls pay UGX. 2000➢ Meat stalls pay UGX. 2500 Use your knowledge of two dimensional and single dimensional arrays to write a c program that findsand displays the following; • Monthly expenditure for each category of stall owner • Annual revenue collected by municiple council from KIKO market. Assuming there are 5 fruit stalls, 10 food stalls and 3meat stalls.

```
#include <stdio.h>


int main() {
    // Part a
    char name[] = "John Doe";


    // Part b
    int results[] = {50, 68, 90};
    int total = 0;
    for (int i = 0; i < sizeof(results) / sizeof(results[0]); ++i) {
        total += results[i];
    }
    results[3] = total / (sizeof(results) / sizeof(results[0]));


    // Part c
    int fruitStalls = 5;
```

```
    int foodStalls = 10;

    int meatStalls = 3;



    // Daily fees

    int fruitFee = 1000;

    int foodFee = 2000;

    int meatFee = 2500;



    // Monthly expenditure

    int monthlyExpenditure = (fruitStalls * fruitFee) + (foodStalls * foodFee) + (meatStalls *
meatFee);



    // Annual revenue

    int daysInMonth = 30; // Assuming a month has 30 days

    int annualRevenue = monthlyExpenditure * daysInMonth;



    // Display results

    printf("a) Name: %s\n", name);



    printf("b) Average of results: %d\n", results[3]);



    printf("c) Monthly expenditure for each category of stall owner:\n");

    printf("    - Fruit stalls: UGX %d\n", fruitStalls * fruitFee);

    printf("    - Food stalls: UGX %d\n", foodStalls * foodFee);

    printf("    - Meat stalls: UGX %d\n", meatStalls * meatFee);
```

```
    printf("    Annual revenue collected by municipal council from KIKO market: UGX %d\n",
annualRevenue);



    return 0;

}
```

What is a function in c programming?

A function in C is a self-contained block of code that performs a specific task. It is designed to be reusable and modular, making the code more organized and easier to understand. Functions help break down complex problems into smaller, manageable pieces.

b) Explain the different parts that make up the structure of a c function.

- **Return Type:** Specifies the type of value the function will return to the calling program. If the function doesn't return any value, the return type is specified as `void`.
- **Function Name:** A unique identifier for the function.
- **Parameters:** Input values that the function can use during its execution. Parameters are optional, and a function can have zero or more parameters.
- **Function Body:** The block of code enclosed in curly braces `{}` that defines what the function does. It contains declarations, statements, and expressions.
- **Return Statement:** If the function has a return type other than `void`, it must contain a `return` statement to send a value back to the calling program.

c) Write a c function that simply displays a phrase of your choice.

```
#include <stdio.h>

void displayPhrase() {

    printf("Hello, C Programming!\n");

}

int main() {

    displayPhrase();

    return 0;

}
```

d) Write a c program containing a main function and at least two additional functions; area which computes the area of a square and perimeter which computes the perimeter of a square. The functions should make use of one parameter called length with a basic assumption that all the sides are the same length. Invoke these functions in the main method and display both the area and perimeter on the screan. Hint: Area = (length * length) Perimeter = (length +length+length+length)

```
#include <stdio.h>

// Function to compute the area of a square

double computeArea(double length) {

    return length * length;

}
```

```
// Function to compute the perimeter of a square
double computePerimeter(double length) {
    return 4 * length;
}
int main() {
    double length;
   // Assuming length is provided by the user or obtained from some source
    printf("Enter the length of the square: ");
    scanf("%lf", &length);
     // Compute and display the area
    double area = computeArea(length);
    printf("Area of the square: %.2f square units\n", area);

    // Compute and display the perimeter
    double perimeter = computePerimeter(length);
    printf("Perimeter of the square: %.2f units\n", perimeter);

    return 0;
}
```

the student guild government of UCU is trying to improve it's decision making process by involving the entire student community. to do this, they want to have an application which allows interaction between the guild and the student community on issues which affect them. the application will be accessed by registered UCU students who will log into the application using their access number and a password. when they successfully login, they will be required to 1.

choose an existing topic of discussion or 2. create a new topic of discussion or 3. submit a complaint. incase the student has chosen to begin a new topic of discussion, after the topic has been posted, a guild official in charge of the communication asseses the topic to determine its validity. if the topic violates the UCU core values, it is deleted. otherwise the guild official opens up the discussion by inviting everyone to give their opinion. write a c program to help with this

```
#include <stdio.h>

#include <string.h>



#define MAX_TOPICS 100

#define MAX_TOPIC_LENGTH 256

#define MAX_COMPLAINT_LENGTH 512

#define MAX_USERS 100



typedef struct {

    int topicID;

    char topic[MAX_TOPIC_LENGTH];

    int isValid;

} Topic;



typedef struct {

    int userID;

    char accessNumber[10];

    char password[20];
```

```
} User;

User users[MAX_USERS];
int userCount = 0;

Topic topics[MAX_TOPICS];
int topicCount = 0;

void registerUser(char accessNumber[], char password[]) {
    User newUser;
    newUser.userID = userCount + 1;
    strcpy(newUser.accessNumber, accessNumber);
    strcpy(newUser.password, password);

    users[userCount++] = newUser;
}

int loginUser(char accessNumber[], char password[]) {
    for (int i = 0; i < userCount; i++) {
        if (strcmp(users[i].accessNumber, accessNumber) == 0 &&
            strcmp(users[i].password, password) == 0) {
            return i; // Return the index of the logged-in user
        }
    }
    return -1; // User not found
```

```
}

void createTopic(int userID, char topic[]) {
    Topic newTopic;
    newTopic.topicID = topicCount + 1;
    strcpy(newTopic.topic, topic);
    newTopic.isValid = 1; // Assume it's valid initially

    // Validate the topic (you can add more sophisticated validation logic)
    if (strstr(topic, "badword") != NULL) {
        newTopic.isValid = 0; // Invalid topic
    }

    topics[topicCount++] = newTopic;
}

void displayTopics() {
    printf("\nAvailable Topics:\n");
    for (int i = 0; i < topicCount; i++) {
        printf("%d. %s\n", topics[i].topicID, topics[i].topic);
    }
}

int main() {
    char accessNumber[10], password[20];
```

```
int userID;

// Simulate user registration
registerUser("123456", "password123");

// Simulate user login
printf("Enter access number: ");
scanf("%s", accessNumber);
printf("Enter password: ");
scanf("%s", password);

userID = loginUser(accessNumber, password);

if (userID == -1) {
    printf("Invalid credentials. Exiting.\n");
    return 1;
}

printf("Login successful!\n");

int choice;
char topic[MAX_TOPIC_LENGTH];

do {
    printf("\n1. Choose an existing topic\n");
```

```
printf("2. Create a new topic\n");
printf("3. Submit a complaint\n");
printf("4. Exit\n");
printf("Enter your choice: ");
scanf("%d", &choice);

switch (choice) {
    case 1:
        displayTopics();
        // Add logic for choosing an existing topic
        break;
    case 2:
        printf("Enter the topic: ");
        getchar(); // Clear newline from buffer
        fgets(topic, sizeof(topic), stdin);
        topic[strlen(topic) - 1] = '\0'; // Remove trailing newline
        createTopic(userID, topic);
        printf("Topic created successfully!\n");
        break;
    case 3:
        // Add logic for submitting a complaint
        break;
    case 4:
        printf("Exiting...\n");
        break;
```

```
            default:
                printf("Invalid choice. Try again.\n");
        }
    } while (choice != 4);

    return 0;
}
```

in January 2023, Michael acquired a loan of 20million Uganda shillings from his friend. he was required to pay 1.5million every month till he completes the full payment. write a c program that will help us to know the number of months it will take Michael to complete payment

```
#include <stdio.h>

int main() {
    // Loan details
    double loanAmount = 20000000.0;  // 20 million Uganda shillings
    double monthlyPayment = 1500000.0;  // 1.5 million Uganda shillings per month

    // Calculate the number of months
    int months = (int)(loanAmount / monthlyPayment);

    // Output the result
    printf("It will take Michael %d months to complete the payment.\n", months);
```

```
    return 0;
}
```

ministry of education has rolled out a scholarship scheme for university students. it is offering to pay 100% for PhD students, 75% for master's students and 50% for Bachelor's students. when an applicant qualifies for a particular bursary, they are asked for their full fees(cost of their course). then a computation is made to establish how much the ministry will be paying for them and they are given feedback to that effect. write a c program to help with this

```
#include <stdio.h>

int main() {
    // Declare variables
    double fullFees, ministryPayment;
    int scholarshipType;

    // Input the full fees and scholarship type
    printf("Enter the full fees for the course: $");
    scanf("%lf", &fullFees);

    printf("Select the scholarship type:\n");
    printf("1. PhD (100%%)\n");
    printf("2. Master's (75%%)\n");
    printf("3. Bachelor's (50%%)\n");
```

```
printf("Enter the corresponding number: ");
scanf("%d", &scholarshipType);

// Validate scholarship type
if (scholarshipType < 1 || scholarshipType > 3) {
    printf("Invalid scholarship type. Please select 1, 2, or 3.\n");
    return 1; // Exit with an error code
}

// Calculate ministry payment based on scholarship type
switch (scholarshipType) {
    case 1:
        ministryPayment = fullFees * 1.0; // 100% for PhD
        break;
    case 2:
        ministryPayment = fullFees * 0.75; // 75% for Master's
        break;
    case 3:
        ministryPayment = fullFees * 0.5; // 50% for Bachelor's
        break;
}

// Output the result
```

```
    printf("Ministry of Education will pay: $%.2lf\n", ministryPayment);


    return 0;
}
```

what security measures need to be considered for any application generally

.

Authentication and Authorization:

.

- Implement strong authentication mechanisms, such as multi-factor authentication (MFA), to ensure that only authorized users can access the application.
- Use appropriate authorization controls to define and enforce access levels for different users or roles.

.

Data Encryption:

.

- Encrypt sensitive data both in transit and at rest to protect it from unauthorized access. Use protocols like HTTPS for communication over the network.
- Utilize strong encryption algorithms and key management practices.

.

Input Validation:

.

- Validate and sanitize all user inputs to prevent injection attacks, such as SQL injection, cross-site scripting (XSS), and cross-site request forgery (CSRF).
- Apply input validation on both the client and server sides.

.

Session Management:

.

- Implement secure session management to protect user sessions from hijacking or session fixation attacks.
- Use secure session protocols and regenerate session identifiers after login.

.

Error Handling:

.

- Provide custom error messages to users and log detailed error information internally. Avoid exposing sensitive information in error messages.
- Implement proper error-handling mechanisms to prevent information disclosure.

.

Security Headers:

.

- Use security headers like Content Security Policy (CSP), Strict-Transport-Security (HSTS), and X-Content-Type-Options to enhance the security of web applications.

.

File Upload Security:

.

- If your application allows file uploads, validate file types, limit file sizes, and store uploaded files in a secure location. Ensure proper input validation for file uploads.

.

Security Patching:

.

- Regularly update and patch the application and its dependencies to address known vulnerabilities. Stay informed about security updates for all components.

.

Firewalls and Intrusion Detection/Prevention Systems (IDS/IPS):

.

- Use firewalls to monitor and control incoming and outgoing traffic. Implement intrusion detection and prevention systems to identify and respond to potential threats.

.

Logging and Monitoring:

.

- Implement comprehensive logging to record security-relevant events. Regularly review logs to detect and respond to security incidents.
- Set up monitoring systems to alert administrators about unusual activities or potential security breaches.

.

Dependency Scanning:

.

- Regularly scan and audit third-party libraries and dependencies for security vulnerabilities. Keep dependencies updated to their latest secure versions.

.

Security Training and Awareness:

.

- Educate development and operational teams about security best practices. Conduct regular security training to raise awareness about potential threats and vulnerabilities.

.

Incident Response Plan:

.

- Develop and document an incident response plan to guide the team in the event of a security incident. Define roles and responsibilities for incident response.

.

Regular Security Audits and Penetration Testing:

.

- Conduct regular security audits and penetration testing to identify and address vulnerabilities proactively. Regular testing helps discover and fix security issues before they can be exploited.

.

Algorithm: A step-by-step procedure or formula for solving a problem. In programming, algorithms are implemented using code.

.
.

Array: A collection of elements, all of the same type, identified by an index or a key.

.
.

Boolean: A data type that can only have two values, typically `true` or `false`. Used in logical expressions and conditions.

.
.

Compiler: A program that translates source code written in a high-level programming language (like C) into machine code or an intermediate code.

.
.

Constant: A value that does not change during the execution of a program. Constants are often used for fixed values in the code.

.
.

Function: A named block of code that performs a specific task. Functions allow for code modularity and reusability.

.
.

Header File: A file containing declarations for functions, variables, and macros that are used in a C program. Common header files include `<stdio.h>` and `<stdlib.h>`.

.
.

Integer: A data type representing whole numbers without a fractional component.

.
.

Loop: A control flow statement that allows a piece of code to be executed repeatedly based on a certain condition.

.
.

Pointer: A variable that stores the memory address of another variable. Pointers are often used for dynamic memory allocation and manipulation.

.
.

Struct: A user-defined data type in C that allows bundling together variables of different types under a single name.

.
.

Syntax: The set of rules that dictate how programs in a specific programming language are constructed.

.
.

Variable: A named storage location in a program that holds a value, which can change during the execution of the program.

.
.

Conditional Statement: A statement that performs different actions depending on whether a condition is true or false. Common examples include `if`, `else if`, and `switch` statements.

.
.

Data Type: A classification that specifies which type of value a variable can hold, such as `int` for integers, `float` for floating-point numbers, and `char` for characters.

.

.

Operator: A symbol that represents a specific operation on one or more operands. Examples include arithmetic operators (`+`, `-`, `*`, `/`) and relational operators (`==`, `!=`, `<`, `>`).

.
.

Pointer Arithmetic: Performing arithmetic operations on pointers, such as incrementing or decrementing a pointer to navigate through the elements of an array.

.
.

Dynamic Memory Allocation: The process of allocating memory during program execution, typically using functions like `malloc()` and `free()`.

.
.

Recursion: A programming technique where a function calls itself in order to solve a problem.

.
.

Header Guard: A preprocessor directive (e.g., `#ifndef`, `#define`, `#endif`) used to prevent multiple inclusions of the same header file in a program.

.