INTRODUCTION TO PROGRAMMING ASSIGNMENT

1. Write short notes for loop with examples of while loop and do…while loop
2. Write notes on arrays, structures, functions and errors a programmer is likely to encounter in C programming
3. Write uses on applications of loops, arrays, structures and union

Use a snippet of code, explain difference between while loop and do… while loop.

1) LOOPS: WHILE LOOP AND DO…WHILE LOOP

a) While Loop

-The `while` loop is a pre-test loop, meaning it checks the condition before entering the loop.

Syntax

```
while (condition) {
    // code to be executed
}
```

Example:

```
#include <stdio.h>
int main() {
    int i = 1;
    while (i <= 5) {
        printf("%d ", i);
        i++;
    }
    return 0;
}
```

Output:

```
1 2 3 4 5
```

b) Do...While Loop:

-The `do...while` loop is a post-test loop, meaning it checks the condition after executing the loop's body.

Syntax:

```
do {
    // code to be executed
} while (condition);
```

Example:

```
#include <stdio.h>
int main() {
    int i = 1;
    do {
        printf("%d ", i);
        i++;
    } while (i <= 5);
    return 0;
}
```

Output:

```
1 2 3 4 5
```

Key Differences:

-In a `while` loop, the condition is checked before the loop body is executed, so it's possible that the loop body may not execute at all.

-In a `do...while` loop, the loop body is executed at least once because the condition is checked after the first execution.

- These loops are fundamental for controlling the flow of a program, especially when you need to repeat a certain block of code multiple times based on a condition.

2) ARRAYS, STRUCTURES, FUNCTIONS AND ERRORS A PROGRAMMER IS LIKELY TO ENCOUNTER IN C PROGRAMMING

a) Arrays:

An array is a collection of elements of the same data type stored in contiguous memory locations.

Declaration:

`datatype arrayName[size];`

Elements are accessed using indices, starting from 0: `arrayName[index]`.

Example:

```
int numbers[5] = {1, 2, 3, 4, 5};

printf("%d", numbers[2]); // Outputs: 3
```

b) Structures:

A structure is a user-defined data type that allows grouping variables of different data types under a single name.

Declaration:

```
struct structName {
    datatype member1;
    datatype member2;
    // ...
};
```

Members are accessed using the dot (`.`) operator: `structName.member`.

Example:

```
struct Point {
    int x;
    int y;
};
struct Point p1 = {10, 20};
printf("%d", p1.x); // Outputs: 10
```

c) Functions:

Functions are blocks of code that perform a specific task. They are modular and promote code reusability.

Declaration:

```
returnType functionName(parameter1Type param1, parameter2Type param2, ...) {
    // Function body
  }
```

Example:

```
int add(int a, int b) {
   return a + b;
}
```

Errors in C Programming:

1. Syntax Errors: Incorrect syntax that prevents the program from compiling.
2. Logical Errors: Program compiles but does not produce the expected result due to flawed logic.
3. Runtime Errors: Errors that occur during program execution, such as division by zero or accessing an invalid memory location.
4. Linker Errors: Occur during the linking phase if functions or variables are not properly defined.
5. Segmentation Fault: Accessing restricted memory areas, often caused by improper use of pointers.
6. Array Out of Bounds: Accessing array elements beyond their defined size.
7. Uninitialized Variables: Using variables without initializing them first.

Tips for Debugging:

1. Print Statements: Use `printf` statements to print variable values and trace program flow.
2. Debugger: Utilize a debugger to step through code and identify issues.
3. Code Reviews: Have peers review your code to catch errors and provide feedback.
4. Compile with Warnings: Enable compiler warnings (`-Wall` flag) to catch potential issues.

3) USES ON APPLICATIONS OF LOOPS, ARRAYS, STRUCTURES & UNIONS

a) Loops:

-Iteration: Loops are primarily used for repetitive execution of a block of code. This is essential when you need to perform a task multiple times without duplicating the code.

-Array/Collection Processing: Loops are commonly used to iterate over elements in an array or other data structures, performing operations on each element.

-Input Validation: Loops can be used to repeatedly prompt users for input until valid data is provided.

b) Arrays:

-Data Storage: Arrays are used to store and organize collections of data of the same type. For example, a list of integers, strings, or custom objects.

-Access and Manipulation: Arrays provide a way to access individual elements using indices, allowing for efficient retrieval and manipulation of data.

-Iteration: Arrays work seamlessly with loops, allowing you to iterate over all elements in the array for various operations.

c) Structures:

-Data Organization: Structures are used to group together different data types under a single name. This is helpful for organizing related pieces of information into a single, cohesive unit.

-Function Parameters: Structures are often used to pass multiple parameters to functions. This improves code readability and maintainability.

-Database Records: In database programming, structures can represent records, where each field corresponds to a column in a database table.

d) Unions:

-Memory Efficiency: Unions allow you to define a structure where different data types share the same memory location. This is useful when you want to save memory by representing the same piece of information in different ways.

-Interpretation of Data: Unions are used when a variable can hold different types of data at different times, and you want to interpret the same memory content differently based on the context.

-File Parsing: Unions can be used when parsing binary files with different data types, allowing you to interpret the same block of memory as different types of data.

In summary, loops, arrays, structures, and unions are fundamental constructs in programming, each serving distinct purposes. They are essential tools for organizing and manipulating data, controlling program flow, and improving code efficiency and readability.

4. CODES EXPLAINING DIFFERENCE BETWEEN WHILE LOOP AND DO…WHILE LOOP

(i) While Loop:

In a `while` loop, the condition is checked before the execution of the loop body. If the condition is false initially, the loop body may not execute at all.

```
#include <stdio.h>
int main() {
    int i = 0;
    // While loop
    while (i < 5) {
        printf("While Loop: %d\n", i);
        i++;
    }
    return 0;
}
```

(ii) <u>Do...While Loop:</u>

```
#include <stdio.h>
int main() {
    int i = 0;
    // Do...While loop
    do {
        printf("Do...While Loop: %d\n", i);
        i++;
    } while (i < 5);
    return 0;
}
```

In summary:

- `while` checks the condition before entering the loop.

- `do...while` checks the condition after executing the loop body, ensuring at least one iteration.