NOTES

Computer Fundamentals and Programming

Unit 2:Introduction to C Programming

•Basics of C Language:

- **Structure of C Program**:

A C program generally follows a specific structure, which includes various components. Here is a basic outline of the structure of a C program:

1. Documentation Section:
- Comments that provide information about the program.
- Comments can be in the form of a header comment at the top of the file explaining the purpose of the program, the author, date, etc.

```
/*
  Program: Name_of_Program
  Author: Your_Name
  Date: Date_of_Creation
*/
```

2. Header Files:
- Include necessary header files for input/output operations and other functionalities.

```
#include <stdio.h>
#include <stdlib.h> // Example of additional header file
```

3. Constants and Macros:
- Define constants and macros that will be used throughout the program.

```
#define PI 3.14159
```

4. Global Declarations:
- Declare global variables and functions that will be used in the program.

```
    int global_variable;

void myFunction();
```

5. main() Function:
- Every C program must have a `main()` function where execution begins.
- The `main()` function can call other functions and may return a value to the operating system.

```
int main() {
    // Code for the main function
    return 0;
}
```

6. Functions:
- Define additional functions below the `main()` function.
- Each function should have its own purpose and be designed to perform a specific task.

```
void myFunction() {
    // Code for the custom function
}
```

7. Variable Declarations:
- Declare local variables within functions.

```
int main() {
    int local_variable;
    // Code for the main function
    return 0;
}
```

8. Statements and Expressions:
- Write the actual code to perform the desired operations.
- Use statements and expressions to manipulate variables and control program flow.

```
int main() {
    int x, y, sum;
```

NOTES

```
    x = 5;
    y = 10;
    sum = x + y;
    printf("Sum: %d\n", sum);
    return 0;
}
```

9. Input/Output Operations:
- Include statements for input and output operations using functions like `printf` and `scanf`.

```
printf("Enter a number: ");
scanf("%d", &x);
```

10. Control Structures:
- Use control structures like `if`, `else`, `while`, `for` to control the flow of the program.

```
if (x > 0) {
    // Code to execute if x is greater than 0
} else {
    // Code to execute if x is less than or equal to 0
}
```

11. Error Handling:
- Implement error handling mechanisms as needed, using constructs like `if`, `else`, and error codes.

```
if (error_condition) {
    // Code to handle the error
}
```

12. Cleanup Code:
- Include any necessary code for releasing resources or cleaning up before the program exits.

```
// Cleanup code
```

NOTES

This is a general structure, and the complexity of your program may determine how closely you adhere to it. However, following a structured approach makes your code more readable and maintainable.

- **Variables:**

In C, a variable is a storage location with a symbolic name and a specific data type. It is used to store data during the execution of a program.

Example:

```
#include <stdio.h>

int main() {
    // Variable declaration
    int age;

    // Variable assignment
    age = 25;

    // Variable usage
    printf("Age: %d\n", age);

    return 0;
}
```

Output:
Age: 25

- **Data Types:**

C supports various data types, including int, float, double, char, etc. Data types specify how the data should be stored and interpreted.

Example:

```
#include <stdio.h>

int main() {
    // Data type declaration
```

```
    int integerVar = 10;
    float floatVar = 3.14;
    char charVar = 'A';

    // Display values
    printf("Integer: %d\n", integerVar);
    printf("Float: %f\n", floatVar);
    printf("Character: %c\n", charVar);

    return 0;
}
```

Output:

```
Integer: 10
Float: 3.140000
Character: A
```

- **Type Casting:**

  Type casting is the conversion of one data type to another. It can be implicit or explicit.

  Example:

```
#include <stdio.h>

int main() {
    int integerVar = 10;
    float floatVar;

    // Implicit type casting
    floatVar = integerVar;

    // Display values
    printf("Integer: %d\n", integerVar);
    printf("Float: %f\n", floatVar);

    return 0;
}
```

Output:
Integer: 10
Float: 10.00000

- **Expressions:**

  Expressions are combinations of operators and operands that, when evaluated, produce a result.

  Example:

```
#include <stdio.h>

int main() {
    // Expressions
    int a = 5, b = 3;
    int sum = a + b;
    int product = a * b;

    // Display results
    printf("Sum: %d\n", sum);
    printf("Product: %d\n", product);

    return 0;
}
```

  Output:
  Sum: 8
  Product: 15

- **Input/Output Operations:**

  C uses `printf` for output and `scanf` for input operations.

  Example:

```
#include <stdio.h>

int main() {
    // Input
    int num;
```

```
    printf("Enter a number: ");
    scanf("%d", &num);

    printf("You entered: %d\n", num);

    return 0;
}
```

Output:

```
Enter a number: 42
You entered: 42
```

Operators in C are symbols that represent computations or operations on variables or values. They perform various tasks, such as arithmetic operations, logical operations, bitwise operations, and more. Here's an overview of some common types of operators in C:

- **Operators in C:**

1. Arithmetic Operators:

These operators perform basic arithmetic operations.

- `+` (Addition): Adds two operands.

```
int sum = a + b;
```

- `-` (Subtraction):Subtracts the right operand from the left operand.

```
int difference = a - b;
```

- `*` (Multiplication): Multiplies two operands.

```
int product = a * b;
```

- `/` (Division): Divides the left operand by the right operand.

```
int quotient = a / b;
```

- `%` (Modulus): Returns the remainder of the division of the left operand by the right operand.

```
int remainder = a % b;
```

2. Relational Operators:

These operators are used to compare values.

- `==` (Equal to): Checks if two operands are equal.

```
if (a == b) {
    // code
}
```

- `!=` (Not equal to): Checks if two operands are not equal.

```
if (a != b) {
    // code
}
```

- `>` (Greater than):Checks if the left operand is greater than the right operand.

```
if (a > b) {
    // code
}
```

- `<` (Less than): Checks if the left operand is less than the right operand.

```
if (a < b) {
    // code
}
```

- `>=` (Greater than or equal to):Checks if the left operand is greater than or equal to the right operand.

```
if (a >= b) {
```

```
    // code
}
```

- `<=` (Less than or equal to):Checks if the left operand is less than or equal to the right operand

```
if (a <= b) {
    // code
}
```

3. Logical Operators:

These operators perform logical operations.

- `&&` (Logical AND): Returns true if both operands are true.

```
if (a > 0 && b > 0) {
    // code
}
```

- `||` (Logical OR):Returns true if at least one of the operands is true.

```
if (a > 0 || b > 0) {
    // code
}
```

- `!` (Logical NOT): Returns true if the operand is false and vice versa.

```
if (!(a > 0)) {
    // code
}
```

4. Assignment Operators:

These operators assign values to variables.

- `=` (Assignment): Assigns the value on the right to the variable on the left.

```
int x = 10;
```

NOTES

- `+=`, `-=`, `*=`, `/=`, `%=` (Compound Assignment): Combines an arithmetic operation with assignment.

```
a += b; // equivalent to a = a + b;
```

5. Bitwise Operators:

These operators perform operations on individual bits.

- `&` (Bitwise AND):
- `|` (Bitwise OR):
- `^` (Bitwise XOR):
- `<<` (Left Shift):
- `>>` (Right Shift):
- `~` (Bitwise NOT):

6. Conditional (Ternary) Operator:

A shorthand way of writing an `if-else` statement.

```
int result = (a > b) ? a : b;
```

These are some of the key operators in C. Understanding how they work is crucial for writing effective and concise C code.

- **Decision Control Statements:**
Decision control statements are used in programming to control the flow of execution based on certain conditions. Here's an explanation of the ones you mentioned:

1. if statement:
- The `if` statement is a basic decision-making statement that allows you to execute a block of code if a specified condition is true.

```
if condition:
    # code to be executed if the condition is true
```

2. if...else statement:

NOTES

- The `if...else` statement extends the `if` statement by providing an alternative block of code to be executed if the condition is false.

```
if condition:
    # code to be executed if the condition is true
else:
    # code to be executed if the condition is false
```

3. nested if-else statement:

- This involves placing an `if...else` statement inside another `if` or `else` block. It allows for more complex decision-making.

```
if condition1:
    # code to be executed if condition1 is true
    if condition2:
        # code to be executed if both condition1 and condition2 are true
    else:
        # code to be executed if condition1 is true but condition2 is false
else:
    # code to be executed if condition1 is false
```

4. cascaded if-else statement:

- This is a series of `if...else` statements where each `else` is followed by another `if` condition. It's used when there are multiple conditions to be checked.

```
if condition1:
    # code to be executed if condition1 is true
elif condition2:
    # code to be executed if condition1 is false and condition2 is true
elif condition3:
    # code to be executed if both condition1 and condition2 are false, and condition3 is true
else:
    # code to be executed if all conditions are false
```

5. switch statement:

- In some programming languages (like C++ or Java), there is a `switch` statement that allows for multi-way branching based on the value of an expression.

NOTES

```
switch (expression) {
 case value1:
    // code to be executed if expression is equal to value1
    break;
 case value2:
    // code to be executed if expression is equal to value2
    break;
 // more cases as needed
 default:
    // code to be executed if none of the cases match
}
```

- **Interview Question :**

1. Explain the difference between the if statement and the if...else statement in C.
2. How does type casting work in C? Provide an example demonstrating the concept of implicit and explicit type casting.
3. Illustrate the use of the ternary operator (? :) in C with a practical example.
4. Define a variable, initialize it with a value, and demonstrate the use of at least two different data types in C. Explain the significance of choosing appropriate data types.
5. Write a simple C function that takes two parameters and returns their sum. Demonstrate how this function can be used in the main() function with appropriate variable declarations and function calls.