PROGRAMMING ASSIGNMENT

1. Create a C program to store and manage student details.
   Registration number, 1st name, 2nd name, last name and units (intro to programming, networking, accounting, OS and Computer Applications)
2. Define the following terms as used in programming:
   a) Compiler
   b) Source code
   c) Object code
   d) Linkers
3. Using an example i.e. a program to add 2 numbers. Explain the compilation process of a C compiler
4. Explain the difference between a compiler and an interpreter. (at least 6 comparisons)
5. List all the main categories of operations available in C programming

Sure, I can help you with that! Let's start with the C program for managing student details:

```c
#include <stdio.h>

struct Student {
    int regNumber;
    char firstName[50];
    char secondName[50];
    char lastName[50];
    int units[5];
};

int main() {
    struct Student student;

    // Input student details
    printf("Enter Registration Number: ");
    scanf("%d", &student.regNumber);

    printf("Enter First Name: ");
    scanf("%s", student.firstName);

    printf("Enter Second Name: ");
    scanf("%s", student.secondName);

    printf("Enter Last Name: ");
    scanf("%s", student.lastName);

    printf("Enter Units (5): ");
    for (int i = 0; i < 5; i++) {
        scanf("%d", &student.units[i]);
    }

    // Display student details
    printf("\nStudent Details:\n");
    printf("Registration Number: %d\n", student.regNumber);
    printf("Name: %s %s %s\n", student.firstName, student.secondName, student.lastName);
    printf("Units: %d, %d, %d, %d, %d\n", student.units[0], student.units[1], student.units[2], student.units[3], student.units[4]);

    return 0;
}
```

Now, let's move on to defining the terms used in programming:

1. **Compiler:**

   - A compiler is a program that translates the entire source code of a high-level programming language into machine code or an intermediate code.

   - It checks the entire program for errors before translation.

   - Examples include GCC (GNU Compiler Collection), Clang, and Microsoft Visual C++ Compiler.

2. **Source Code:**

   - Source code is the human-readable, high-level representation of a program written in a programming language.

   - It is the input to a compiler or interpreter.

   - It consists of statements and instructions that define the logic and behavior of a program.

3. **Object Code:**

   - Object code is the output generated by a compiler after translating the source code.

   - It is in a low-level, machine-readable format but is not yet the final executable.

   - Object code is specific to the target architecture for which it was compiled.

4. **Linkers:**

   - Linkers are tools that combine multiple object files into a single executable program.

   - They resolve references between different files, ensuring that functions and variables are correctly linked.

   - Linkers create the final executable by combining object code and resolving addresses.

Now, let's explain the compilation process using an example program to add two numbers:

```c
#include <stdio.h>


int main() {
    int num1, num2, sum;


    printf("Enter two numbers: ");
    scanf("%d %d", &num1, &num2);


    sum = num1 + num2;


    printf("Sum: %d\n", sum);


    return 0;
}
```

Compilation Process:

1. **Preprocessing:** The preprocessor handles include statements and macros. It produces an intermediate code.

2. **Compilation:** The compiler translates the preprocessed code into assembly code or machine code.

3. **Assembly:** The assembler converts the assembly code into object code.

4. **Linking:** The linker combines object code with other necessary code (like libraries) to create the final executable.

Now, let's explain the differences between a compiler and an interpreter:

1. **Execution:**

   - Compiler: Produces an executable file, and the entire program is translated before execution.

   - Interpreter: Translates and executes the source code line by line.

2. **Speed:**

   - Compiler: Generally produces faster code as the entire program is optimized before execution.

   - Interpreter: May have a slower execution as it translates code on-the-fly.

3. **Platform Dependency:**

   - Compiler: Generates platform-specific executables, requiring recompilation for different platforms.

   - Interpreter: Generally platform-independent, as the interpreter itself is platform-specific.

4. **Error Detection:**

   - Compiler: Detects errors for the entire program before execution.

   - Interpreter: Halts at the first encountered error, making it easier to debug.

5. **Memory Usage:**

   - Compiler: Usually results in a standalone executable file, potentially using more memory.

   - Interpreter: Typically requires less memory but needs the interpreter to be present.

6. **Portability:**

   - Compiler: Less portable, as different platforms may require recompilation.

   - Interpreter: More portable, as long as the interpreter is available for the target platform.

Finally, let's list the main categories of operations available in C programming:

1. **Arithmetic Operations:**
   - Addition, subtraction, multiplication, division, modulus.

2. **Relational Operations:**
   - Equality, inequality, greater than, less than, greater than or equal to, less than or equal to.

3. **Logical Operations:**
   - AND, OR, NOT.

4. **Bitwise Operations:**
   - Bitwise AND, OR, XOR, left shift, right shift.

5. **Assignment Operations:**
   - Simple assignment, compound assignment (e.g., `+=`, `-=`, `*=`, `/=`).

6. **Increment and Decrement Operations:**
   - Increment (`++`), decrement (`--`).

7. **Conditional (Ternary) Operation:**
   - `condition ? expression_if_true : expression_if_false`.

8. **Control Flow Operations:**
   - `if`, `else`, `switch`, `while`, `do-while`, `for`.

9. **Function Call and Return Operations:**
   - Defining functions, calling functions, returning values from functions.

10. **Array and Pointer Operations:**

    - Indexing arrays, pointer arithmetic, dereferencing pointers.

These categories cover a broad range of operations that can be performed in C programming.