Name: Noaman Ayub
Roll # 2k23-BSSE-225
Section: B

Turbo C++ (which was commonly used for **C programming** in the 1990s) supports the **C89** standard (also known as ANSI C). While this version was widely used, it is now outdated and lacks many features that have been added in later versions of the C standard, such as **C99**, **C11**, and **C18**. Below is a detailed list of key differences between **Turbo C** (C89) and modern versions of the **C standard** (C99 and later):

Key Features Not Available in Turbo C (C89)

1. Modifying String Literals (const correctness)

- **Turbo C (C89)** allows modifying string literals (e.g., `char* p = "NFC"; *p = 'K';`), but this is **undefined behavior**.
    - In modern compilers (C99 and beyond), string literals are **immutable** and modifying them is not allowed. They are stored in read-only memory, and attempts to change them will cause a runtime error or a segmentation fault.
    - **C99 and beyond**: String literals are treated as `const` (`const char*`).

2. Variable-Length Arrays (VLA)

- **Turbo C (C89)** does **not support Variable Length Arrays** (VLAs), meaning you cannot use variables for array sizes.
    - Example (unsupported in Turbo C):
    - `int n;`
    - `scanf("%d", &n);`
    - `int arr[n];  // Invalid in Turbo C`
- **C99 and beyond**: You can use variables to determine the size of an array at runtime (VLAs are allowed in C99 and later versions).

3. Function Prototypes

- **Turbo C (C89)** does not enforce strict function prototypes, meaning you could call functions before declaring them or even omit the prototype.
    - **C99 and beyond**: Strict function prototypes are required, and function declarations must match their definitions in both parameters and return types.

4. Mixing Declarations and Code

- **Turbo C (C89)** requires that **all variable declarations** be at the **beginning** of a block (before any executable code).
    - **C99 and beyond**: You can declare variables anywhere in a function, even inside loops or conditional blocks.

```
if (x) {
    int a = 5;  // Valid in C99
}
```

5. For Loop Declaration

- **Turbo C (C89)** does not allow **declarations inside the `for` loop**.

```
for (int i = 0; i < 10; i++) {  // Invalid in Turbo C
    // Do something
```

- `}`
- **C99 and beyond**: Variables can be declared directly in the loop statement.

6. Inline Functions

- **Turbo C (C89)** does not support **inline functions**.
  - **C99 and beyond**: You can define **inline functions** using the `inline` keyword, which helps optimize performance by suggesting to the compiler to insert the function's code at the call site rather than performing a function call.
- `inline int square(int x) { return x * x; }`

7. Designated Initializers

- **Turbo C (C89)** does not support **designated initializers** for arrays and structures.
- `int arr[5] = { [0] = 1, [4] = 5 };  // Invalid in Turbo C`
- **C99 and beyond**: Designated initializers are allowed, enabling more flexible initialization of arrays and structs.

8. Flexible Array Members

- **Turbo C (C89)** does not support **flexible array members** in structures.
- `struct s {`
- `    int length;`
- `    int arr[];  // Invalid in Turbo C`
- `};`
- **C99 and beyond**: Flexible array members are allowed in structs, where the last member can be an incomplete array.

9. Standard Library Changes

- **Turbo C (C89)** lacks some modern **standard library functions** that are included in **C99 and beyond**. For example:
  - `stdbool.h` for `bool` data type.
  - `inttypes.h` for `int32_t`, `int64_t` types and related macros.
  - `stdint.h` for fixed-width integer types.

10. Complex Data Types

- **Turbo C (C89)** does not support the **complex number type**.
  - **C99 and beyond**: The `<complex.h>` library and `complex` data type were introduced for handling complex numbers.
- `#include <complex.h>`
- `double complex z = 1.0 + 2.0 * I;`

11. Long Long Integer Type

- **Turbo C (C89)** does not have support for the `long long int` type, which is required for integers larger than `long`.
  - **C99 and beyond**: `long long` and `long long int` types are supported, which allow storing larger integer values (typically 64-bit).

12. Restrict Keyword

- **Turbo C (C89)** does not support the **`restrict` keyword**.

    - **C99 and beyond**: The `restrict` keyword is used to indicate that a pointer is the only reference to the object it points to, which helps the compiler optimize memory access.
- `void foo(int* restrict ptr);`

13. Support for `volatile`

- **Turbo C (C89)** has limited or inconsistent support for the **`volatile`** keyword in certain optimizations or embedded systems programming.
    - **C99 and beyond**: Full and standardized support for `volatile`, used to indicate that a variable's value may change unexpectedly (e.g., due to external hardware).

14. Preprocessor Improvements

- **Turbo C (C89)** has a more limited preprocessor, lacking the more modern macro capabilities.
    - **C99 and beyond**: Introduced features like:
        - `##` for token pasting.
        - `__VA_ARGS__` for variadic macros.

15. Function-like Macros

- **Turbo C (C89)** has limited support for **function-like macros** and preprocessor functionality.
    - **C99 and beyond**: Improved capabilities with variadic macros and function-like macros with variable numbers of arguments.

Additional features and limitations of **Turbo C (C89)** compared to **C99** and later versions that you should be aware of. Below is a more comprehensive list of **features not allowed or limited in Turbo C (C89)**:

1. Missing `stdbool.h` (Boolean Type)

- **Turbo C (C89)** does not have the `<stdbool.h>` header, meaning the `bool` type is not available.
- In **C99** and later, you can use the `bool` type for boolean values instead of using integers (`0` for `false`, `1` for `true`).

```
#include <stdbool.h>
bool isEven = true;
```

2. No `restrict` Keyword

- **Turbo C (C89)** does not support the **`restrict`** keyword, which is used to optimize pointers in C.
- **C99** and later introduce the `restrict` keyword, which informs the compiler that a pointer is the only reference to the object it points to, enabling certain optimizations.

```
void foo(int* restrict ptr);
```

3. Lack of `inttypes.h` for Fixed-width Integer Types

- **Turbo C (C89)** does not include the **`inttypes.h`** header, which provides macros to work with fixed-width integer types (e.g., `int32_t`, `int64_t`).
- In **C99** and beyond, you can use `inttypes.h` for better control over integer sizes.

```
#include <inttypes.h>
int32_t x = 100;
```

4. No Support for `stdint.h` (Standard Integer Types)

- **Turbo C (C89)** does not support **`stdint.h`**, which provides standard integer types (`int32_t`, `int64_t`, etc.) and useful macros like `INT_MAX`.
- In **C99** and beyond, `stdint.h` standardizes integer sizes and improves portability.

```
#include <stdint.h>
int32_t num = 10;
```

5. No `inline` Functions

- **Turbo C (C89)** does not have support for **`inline`** functions, which help optimize performance by replacing function calls with the actual function code in certain cases.
- In **C99** and beyond, the `inline` keyword allows functions to be inlined.

```
inline int square(int x) { return x * x; }
```

6. No Support for `long long` Data Type

- **Turbo C (C89)** does not support the **`long long`** integer type, which is used to store larger integers (typically 64-bit integers).
- In **C99** and beyond, `long long` provides a way to handle larger integer values.

```
long long int largeNumber = 1234567890123456789LL;
```

7. Lack of Complex Numbers

- **Turbo C (C89)** does not have support for **complex numbers** or the `<complex.h>` library.
- In **C99** and beyond, you can work with complex numbers using the `<complex.h>` header.

```
#include <complex.h>
double complex z = 1.0 + 2.0 * I;
```

8. No Support for Designated Initializers

- **Turbo C (C89)** does not support **designated initializers**, which allow initializing specific elements of an array or structure.

```
// Invalid in Turbo C
int arr[5] = { [0] = 10, [4] = 20 };
```

- In **C99** and later, you can use designated initializers for more flexible initialization:

```
int arr[5] = { [0] = 10, [4] = 20 };
```

9. Limited Preprocessor Functionality

- **Turbo C (C89)** has a more **limited preprocessor** with fewer features compared to modern versions.
- **C99** and later support **variadic macros**, allowing macros to accept a variable number of arguments.

```
#define PRINT(fmt, ...) printf(fmt, __VA_ARGS__)
```

10. No Support for Flexible Array Members

- **Turbo C (C89)** does not support **flexible array members** in structures, which is a feature introduced in **C99**.

- **C99** allows defining arrays with a flexible size at the end of structures:

```
struct s {
    int length;
    int arr[];  // Flexible array member (valid in C99)
};
```

11. No `_Bool` Type

- **Turbo C (C89)** does not support the **`_Bool`** type, which was introduced in **C99** as a built-in Boolean type.
- In **C99** and beyond, the `_Bool` type is available, which is distinct from integers used for boolean values in Turbo C.

```
_Bool isValid = 1;
```

12. For Loop Declarations

- **Turbo C (C89)** does not allow **declarations inside the `for` loop**. Declarations must be done before any code in a block.

```
for (int i = 0; i < 10; i++) {  // Invalid in Turbo C
    // Code
}
```

- In **C99** and beyond, you can declare variables within the `for` loop itself:

```
for (int i = 0; i < 10; i++) {  // Valid in C99
    // Code
}
```

13. No `volatile` Keyword (or Limited Support)

- **Turbo C (C89)** has limited or **inconsistent support** for the **`volatile`** keyword in certain cases (such as optimizations in embedded systems).
- **C99** and later versions have **full support** for the `volatile` keyword, which tells the compiler that a variable's value can be changed by external factors (e.g., hardware).

```
volatile int x;  // Valid in C99 and later
```

14. No `#pragma` Directives for Compiler-Specific Features

- **Turbo C (C89)** has limited or no support for many **`#pragma`** directives that modern compilers use to enable specific features or optimizations (e.g., `#pragma pack`, `#pragma once`).
- **C99 and beyond** support various `#pragma` directives for better control over compiler behavior and optimizations.

15. No `_Alignof` (Alignment of Types)

- **Turbo C (C89)** does not support **`_Alignof`** (or **`alignof`** in C11), which is used to check the alignment requirement of a type.

```
size_t alignment = _Alignof(int);  // Valid in C11
```

For modern C development, it is recommended to use a more recent compiler (such as **GCC**, **Clang**, or **MSVC**) and enable **C99** or **C11** standards to take advantage of these features. Turbo C (C89) is now considered outdated and has many limitations in both functionality and safety.