DATE: LAB REPORT NO.: **3** SET:
TITLE OF THE PROGRAM: **STORAGE CLASS**

OBJECTIVES

I. To understand and demonstrate the use of various storage classes in the C programming language, namely:
   1. Auto
   2. Static
   3. Register
   4. Extern

REQUIREMENTS

1. C Compiler (e.g., GCC)
2. Computer System
3. IDE or Text Editor
4. OS compatible with the software

THEORY

Storage classes in C determine the scope, lifetime, and visibility of variables within a program. They are used to allocate memory for variables and define their behavior. The following table gives the information about each storage class:

Class	Name of Class	Place of Storage	Scope	Default Value	Lifetime
auto	Automatic	RAM	Local	Garbage Value	Within a function.
extern	External	RAM	Global	Zero	Till the main program ends.
static	Static	RAM	Local	Zero	Till the main program ends.
register	Register	Register	Local	Garbage Value	Within the function.

PROCEDURE (Program Code, Comment, and Output)

1. **auto storage class**

Program Code:

```c
#include <stdio.h>

void incrementAuto(void);

int main()
{
    incrementAuto();
```

```
    incrementAuto();
    incrementAuto();

    return 0;
}

void incrementAuto(void)
{
    // Declare an auto (automatic storage) integer variable
    auto int count = 0;

    count++;

    printf("Auto count: %d\n", count);
}
```

Output:

```
Auto count: 1
Auto count: 1
Auto count: 1
```

Explanation:

The program includes the standard I/O library `stdio.h` to use the `printf` function for output.
The program defines a function prototype `void incrementAuto(void);` before the `main` function. This prototype tells the compiler about the existence of the `incrementAuto` function, so it can be called from `main` even though the function's actual definition comes later.
The `main` function is the entry point of the program. It calls the `incrementAuto` function three times sequentially.
The `incrementAuto` function is defined after the `main` function. This function demonstrates the behavior of an auto variable, which has automatic storage duration.
Inside the `incrementAuto` function, an auto integer variable `count` is declared and initialized to 0. The keyword `auto` is optional in modern C, as all local variables have automatic storage duration by default.
The value of `count` is then incremented by 1 with the `count++` statement.
The current value of `count` is printed using `printf` with the format string `"Auto count: %d\n"`.
When the `incrementAuto` function is called from `main`, a new instance of the `count` variable is created on the stack. Each call to `incrementAuto` initializes its own `count` variable to 0 and increments it by 1, but as these variables have automatic storage duration, they are destroyed after the function returns. Thus, every time the function is called, it starts with a fresh `count` value of 0, and the output is always `Auto count: 1` three times.

2. **static storage class**

Program Code:

```c
#include <stdio.h>

void incrementStatic(void);

int main()
{
    incrementStatic();
    incrementStatic();
    incrementStatic();

    return 0;
}

void incrementStatic(void)
{
    // Declare and initialize a static variable 'count'
    static int count = 0;

    count++;

    printf("Static count: %d\n", count);
}
```

Output:

```
Static count: 1
Static count: 2
Static count: 3
```

Explanation:

The program defines a function `incrementStatic()` that increments a static variable `count` and prints its value.

In the `main()` function, the `incrementStatic()` function is called three times successively.

The `incrementStatic()` function maintains a static variable `count`, which means that the variable retains its value between function calls. When `incrementStatic()` is called for the first time, `count` is initialized to 0. Then, with each subsequent call, the value of `count` is incremented by 1.

After the three calls to `incrementStatic()`, the output shows the current value of `count` after each call.

3. **register storage class**

Program Code:

```c
#include <stdio.h>
```

```c
int main()
{
    // Declare a register variable 'i' to hold the loop index. The
'register' keyword suggests the compiler to store this variable in a
CPU register for faster access.
    register int i;

    // Declare and initialize a register variable 'sum' to store the
sum of numbers.
    register int sum = 0;

    for (i = 1; i <= 10; i++)
    {
        sum += i;
    }

    // Print the final value of 'sum' using the register variable.
    printf("Sum using register variable: %d\n", sum);

    return 0;
}
```

Output:

```
Sum using register variable: 55
```

Explanation:

The program starts by including the necessary header file `stdio.h`, which allows us to use standard input-output functions like `printf`.

Two variables are declared as `register int`, namely `i` and `sum`. The `register` keyword is used to suggest to the compiler to store these variables in CPU registers if possible, which can lead to faster access during program execution.

The for loop is used to iterate from `i = 1` to `i = 10`. During each iteration, the value of `i` is added to the `sum` variable.

After the loop finishes execution, the value of `sum` will be the sum of numbers from 1 to 10.

Finally, the program prints the calculated sum using the `printf` function.

Note: The use of the `register` keyword is optional, and the compiler may choose to ignore it. In modern compilers, optimizations are performed automatically, so explicitly using `register` might not always have a noticeable impact on performance.

4. **extern storage class**

Program Code:

```c
// File: file1.c

#include <stdio.h>
```

```c
#include "file2.c"    // Include the contents of file2.c

extern int externVariable;    // Declare the external variable defined in file2.c

int main()
{
    printf("Extern variable value: %d", externVariable);    // Print the value of the externVariable

    return 0;
}
```

```c
// File: file2.c
int externVariable = 42;    // Define the external variable and initialize it with the value 42
```

Output:

```
Extern variable value: 42
```

Explanation:

`file1.c` includes the standard input-output library `stdio.h`, which provides functions like `printf` for printing output to the console. It includes the contents of `file2.c` using the preprocessor directive `#include "file2.c"`. This effectively merges the contents of `file2.c` into `file1.c`. The line `extern int externVariable;` is a declaration that informs the compiler about the existence of an integer variable named `externVariable`, which is defined externally (in `file2.c`). In the `main` function, `printf("%d", externVariable);` is used to print the value of `externVariable`. In `file2.c`, the variable `externVariable` is defined and initialized with the value `42`.

CONCLUSION

The project showcased the successful utilization of different storage classes in C programming, including auto, static, register, and extern. Gaining a grasp of these classes enables memory optimization and enhances program performance.