Chapter 4 : Function in C

5.1 Concept of library functions: Library functions are pre-defined functions provided by the C standard library that perform common tasks. These functions are already implemented and can be used by programmers to accomplish specific tasks without having to write the code from scratch.
Example:

```c
#include <stdio.h>

int main() {
    int num = 5;

    // Using the library function printf() to print the value of num
    printf("The value of num is: %d\n", num);

    return 0;
}
```

5.2 String functions (comparison, concatenation, length): String functions are library functions that operate on strings, which are arrays of characters, to perform various operations such as comparison, concatenation, and finding the length of a string.
Examples:

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

int main() {
    char str1[] = "Hello";
    char str2[] = "World";

    // String comparison
    if (strcmp(str1, str2) == 0) {
        printf("The strings are equal.\n");
    } else {
        printf("The strings are not equal.\n");
    }

    // String concatenation
    strcat(str1, " ");
    strcat(str1, str2);
    printf("Concatenated string: %s\n", str1);

    // String length
    int length = strlen(str1);
    printf("Length of the string: %d\n", length);

    return 0;
}
```

5.3 User-defined functions: User-defined functions are functions created by the programmer to perform specific tasks. These functions are defined with a return

type, name, and optional parameters, and can be called from other parts of the program.
Example:
```c
#include <stdio.h>

// User-defined function to calculate the square of a number
int square(int num) {
    return num * num;
}

int main() {
    int num = 5;
    int result = square(num);
    printf("The square of %d is: %d\n", num, result);

    return 0;
}
```

5.4 Local & global variables: Local variables are declared inside a function and can only be accessed within that function. They have local scope, which means they are only visible and usable within the block of code where they are defined.
Global variables, on the other hand, are declared outside of any function and can be accessed by any function in the program. They have global scope, meaning they are visible and can be used throughout the entire program.
Examples:
```c
#include <stdio.h>

// Global variable
int globalVar = 10;

void myFunction() {
    // Local variable
    int localVar = 20;

    printf("Local variable: %d\n", localVar);
    printf("Global variable: %d\n", globalVar);
}

int main() {
    myFunction();

    return 0;
}
```

5.5 Parameter passing: Parameter passing is the mechanism through which values are passed to functions. In C, parameters can be passed to functions either by value or by reference.
Passing by value means that a copy of the value is passed to the function, and any modifications made to the parameter within the function do not affect the original value.

Passing by reference, also known as passing by pointer, means that the address of the variable is passed to the function, allowing the function to directly modify the original value.
Example:

```c
#include <stdio.h>

void changeValue(int *num) {
    *num = 10;
}

int main() {
    int num = 5;
    printf("Before function call: %d\n", num);

    changeValue(&num);
    printf("After function call: %d\n", num);

    return 0;
}
```

5.6 Storage classes: Storage classes in C define the scope, visibility, and lifetime of variables. There are four storage classes in C: auto, static, extern, and register.

- `auto`: The default storage class for local variables. It is automatically allocated and deallocated when the function is called and returns.
- `static`: Allows a local variable to retain its value between function calls. It is allocated once and retains its value throughout the program's execution.
- `extern`: Declares a variable that is defined in another source file. It is used to provide access to a global variable across multiple source files.
- `register`: Requests the compiler to store a variable in a CPU register for faster access. It is a hint to the compiler and may or may not be honored.

Example:

```c
#include <stdio.h>

static int staticVar = 10;
extern int externVar;

int main() {
    auto int localVar = 20;
    register int registerVar = 30;

    printf("Static variable: %d\n", staticVar);
    printf("Extern variable: %d\n", externVar);
    printf("Local variable: %d\n", localVar);
    printf("Register variable: %d\n", registerVar);

    return 0;
}
```

In this example, `staticVar` is a static variable, `externVar` is an external variable defined in another source file, `localVar` is an auto variable, and `registerVar` is a register variable.