What do you mean by one-dimensional array and multi-dimensional array? How arrays are declared and initialized? Give examples

Arrays are data structures that can store multiple values of the same data type under a single variable name. There are two main types of arrays: one-dimensional arrays and multi-dimensional arrays.

1. **One-Dimensional Array:**
    - A one-dimensional array is a collection of elements stored in a contiguous memory location.
    - It is like a list of variables of the same type.

Declaration and Initialization:

```
// Declaration
data_type array_name[size];
// Initialization
data_type array_name[size] = {element1, element2, ..., elementN};
```

Example in C:

```
#include <stdio.h>
#include<conio.h>
int main() {
   // Declaration and Initialization
   int numbers[5] = {1, 2, 3, 4, 5};
   // Accessing elements
   for (int i = 0; i < 5; i++) {
      printf("%d ", numbers[i]);
   }
   return 0;
getch();
}
```

Output:

12345

Multi-**Dimensional Array:**

- A multi-dimensional array is an array of arrays. It can be thought of as a table or matrix with rows and columns.
- Commonly used are 2D arrays, which have rows and columns.

Declaration and Initialization:

```
// Declaration
data_type array_name[rows][columns];

// Initialization
data_type array_name[rows][columns] = {{element11, element12, ..., element1N},
                                       {element21, element22, ..., element2N},
                                       ...,
                                       {elementM1, elementM2, ..., elementMN}};
```

Example in C:

```
#include <stdio.h>
#include<conio.h>
int main() {
   // Declaration and Initialization
   int matrix[3][3] = {{1, 2, 3},
                       {4, 5, 6},
                       {7, 8, 9}};
   // Accessing elements
   for (int i = 0; i < 3; i++) {
      for (int j = 0; j < 3; j++) {
         printf("%d ", matrix[i][j]);
      }
      printf("\n");
   }
   return 0;
getch();
}
Output:
123
456
789
```

These examples are in C, but the concepts are applicable to other programming languages as well. The key idea is that one-dimensional arrays are like lists, and multi-dimensional arrays are like tables or matrices. You can adapt the syntax based on the programming language you are using.

What are the storage classes in c? How these storage classes declared and accessed? with examples and programs

C, there are four storage classes:

1. **auto:**
    - The auto keyword is the default storage class for all local variables.
    - It automatically allocates and deallocates memory for variables.

Example:

```
#include <stdio.h>
#include<conio.h>
void main() {
   auto int x = 10;
   clrscr();
   printf("Value of x: %d\n", x);
   getch();
}
```

Output:

Value of x: 10

register:

- The register keyword is used to suggest to the compiler that the variable should be stored in a register for faster access.
- Like in C++, its usage has diminished due to modern compilers being efficient in register allocation.

Example:

```
#include <stdio.h>
#include<conio.h>
void main() {
   register int counter = 0;
   clrscr();
   for (int i = 0; i < 10; ++i) {
      counter += i;
   }

   printf("Counter: %d\n", counter);
```

```
    getch();
}
```

Output:

Counter: 45

static:

- The static keyword is used for both local and global variables.
- For local variables, it retains its value between function calls.
- For global variables, it restricts the variable's scope to the file where it is declared.

Example:

```
#include <stdio.h>
#include<conio.h>
void demo() {
    static int count = 0;  // Static local variable
    count++;
    printf("Count: %d\n", count);
}
void main() {
    clrscr();
    for (int i = 0; i < 5; ++i) {
        demo();
    }
    getch();
}
```

Output:

Count: 1
Count: 2
Count: 3
Count: 4
Count: 5

extern:

- The extern keyword is used to declare a variable or function that is defined in another file or at some other place in the same file.
- It is often used for global variables.

Example:

```
    // File1.c
```

```
int globalVar = 42;
// File2.c
#include <stdio.h>
#include<conio.h>
extern int globalVar;  // Declaration
void main() {
        clrscr();
        printf("Global Variable: %d\n", globalVar);
        getch();
        return 0;
}
```

Output:

Global Variable: 42

These storage classes control the scope, visibility, and lifetime of variables in C. The choice of storage class depends on the requirements of the program.

What is a file? What type of file you can create in C? Explain with examples how fopen, fclose, fread and fwrite functions are used in files.

In programming, a file is a collection of data stored on a disk with a specific name and a directory path. Files can be used to store various types of data, including text, images, audio, and more. In C, file operations are typically performed using the Standard I/O Library functions.

Here are the basic types of files you can create in C:

1. **Text File:**
    - Stores data in a human-readable format.
    - Examples include .txt files.
2. **Binary File:**
    - Stores data in a binary format, which is not human-readable.
    - Examples include image files, executable files, etc.

Now, let's discuss how to use fopen, fclose, fread, and fwrite functions in C to work with files.

fopen Function:

The fopen function is used to open a file. It takes two arguments - the name of the file and the mode in which the file should be opened (read, write, append, etc.).

```
FILE *fptr;
```

fptr = fopen("filename.txt", "mode");

fclose Function:

The fclose function is used to close an opened file. It is a good practice to close a file after performing operations on it.

fclose(fptr);

fread Function:

The fread function is used to read data from a file. It takes four arguments - a pointer to the memory block where data will be stored, the size of each element to be read, the number of elements to read, and the file pointer.

size_t fread(void *ptr, size_t size, size_t nmemb, FILE *stream);

fwrite Function:

The fwrite function is used to write data to a file. It takes four arguments - a pointer to the data to be written, the size of each element to be written, the number of elements to write, and the file pointer.

size_t fwrite(const void *ptr, size_t size, size_t nmemb, FILE *stream);

Now, let's put it all together with an example:

```
#include <stdio.h>
#include <conio.h>
void main() {
   FILE *file;
   char data[] = "Hello, File Handling!";

   // Writing to a text file
   file = fopen("example.txt", "w");
   if (file == NULL) {
      printf("Error opening file.\n");
      return 1;
   }

   fwrite(data, sizeof(char), sizeof(data), file);
```

```
    fclose(file);

    // Reading from the text file
    char buffer[100];
    file = fopen("example.txt", "r");
    if (file == NULL) {
        printf("Error opening file.\n");
        return 1;
    }

    fread(buffer, sizeof(char), sizeof(data), file);

    fclose(file);

    // Displaying the read data
    printf("Data read from file: %s\n", buffer);

    getch();
}
```

This program writes the string "Hello, File Handling!" to a text file named "example.txt", then reads it back and prints the read data. Remember to handle errors appropriately when working with files in a real-world scenario.

What is string? Explain various functions of string with declaration and access

In C programming, a string is an array of characters that is terminated by a null character '\0'. Strings in C are represented as character arrays, and various functions are available to perform operations on strings.

Here are some commonly used string functions in C:

1. **strlen:**

   Calculates the length of a string.

```
#include <stdio.h>
#include <string.h>
#include<conio.h>
void main() {
        char str[] = "Hello, World!";
        int length = strlen(str);
        clrscr();
        printf("Length of the string: %d\n", length);
        getch();
}
```

Output:

Length of the string: 13

2. **strcpy:**

Copies the content of one string to another.

```
#include <stdio.h>
#include <string.h>
#include<conio.h>
    void main() {
        char source[] = "Hello, World!";
        char destination[20];
        clrscr();
        strcpy(destination, source);
        printf("Source: %s\n", source);
        printf("Destination: %s\n", destination);
        getch();
    }
```

Output:

```
Source: Hello, World!
Destination: Hello, World!
```

3. **strcat:**

Concatenates (appends) one string to the end of another.

```
#include <stdio.h>
#include <string.h>
#include <conio.h>
    void main() {
    char str1[] = "Hello, ";
    char str2[] = "World!";
    char result[20];
   clrscr();
    strcpy(result, str1);
    strcat(result, str2);
    printf("Result: %s\n", result);
    getch();
    }
```

Output:

Result: Hello, World!

4. **strcmp:**

Compares two strings.

```
#include <stdio.h>
#include <string.h>
#include<conio.h>
    void main() {
    char str1[] = "apple";
    char str2[] = "banana";
    clrscr();
    int result = strcmp(str1, str2);
    if (result == 0) {
            printf("Strings are equal.\n");
    } else if (result < 0) {
            printf("String 1 is less than String 2.\n");
    } else {
```

```
        printf("String 1 is greater than String 2.\n");
    }

    getch();
    }
```

Output

```
String 1 is less than String 2.
```

5. **strchr:**

Finds the first occurrence of a character in a string.

```
#include <stdio.h>
#include <string.h>
#include<conio.h>
void main() {
        char str[] = "Hello, World!";
        char *ptr = strchr(str, 'o');
        clrscr();
        printf("First occurrence of 'o' is at position: %ld\n", ptr - str + 1);
        getch();
}
```

Output:

```
First occurrence of 'o' is at position: 5
```

6. **strstr:**

Finds the first occurrence of a substring in a string.

```
#include <stdio.h>
#include <string.h>
#include<conio.h>
void main() {
        char str[] = "Hello, World!";
        char *ptr = strstr(str, "World");
        clrscr();
        if (ptr != NULL) {
                printf("Substring 'World' found at position: %ld\n", ptr - str + 1);
        } else {
                printf("Substring not found.\n");
```

```
        }
    getch();
    }
```

Output:

Substring 'World' found at position: 8

These are just a few examples of the many string functions available in C. Remember to include the <string.h> header to use these functions. Additionally, it's important to be cautious about buffer overflows when working with strings to avoid unexpected behavior in your programs.

What is function? How function declared and what is call by value and call by reference in function Explain with examples and programs

In programming, a function is a self-contained block of code that performs a specific task. Functions provide modularity and allow code to be organized into manageable and reusable units. In C, a function is declared with a return type, a function name, parameters (if any), and a body containing the code to be executed.

Function Declaration:

```
return_type function_name(parameter_list) {
   // function body
}
```

- return_type: The data type of the value the function returns (e.g., int, void).
- function_name: The name of the function.
- parameter_list: The input parameters the function takes (if any).

Call by Value:

In Call by Value, the values of the actual parameters are passed to the function. Changes made to the parameters inside the function do not affect the original values outside the function.

```
#include <stdio.h>
#include<conio.h>
// Function prototype
void increment(int x);
void main() {
   int num = 5;
```

```
clrscr();
printf("Before function call: %d\n", num);
   // Call the function
   increment(num);
   printf("After function call: %d\n", num);
   getch();
}

// Function definition
void increment(int x) {
   x++;
   printf("Inside function: %d\n", x);
}
```

Output:

```
Before function call: 5
Inside function: 6
After function call: 5
```

Call by Reference:

In Call by Reference, the memory address of the actual parameters (pointers) is passed to the function. Changes made to the parameters inside the function affect the original values outside the function.

```
#include <stdio.h>
#include<conio.h>
// Function prototype
void increment(int *x);
void main() {
   int num = 5;
   printf("Before function call: %d\n", num);
   // Call the function
   increment(&num);
   printf("After function call: %d\n", num);
   getch();
}
// Function definition
void increment(int *x) {
```

```
    (*x)++;
    printf("Inside function: %d\n", *x);
}
```

Output:

```
Before function call: 5
Inside function: 6
After function call: 6
```

In the second example (Call by Reference), the function increment takes a pointer as a parameter, and changes made to the value pointed to by the pointer inside the function affect the original variable outside the function.

It's important to note that when working with Call by Reference, you should be careful with pointer manipulations to avoid unintended side effects.