Keywords in C
Keywords are reserved words in the C language that have special meanings and are part of the language's syntax. You cannot use them as identifiers (like variable names) because they are used to perform specific operations. Examples include `int`, `return`, `if`, `else`, `for`, etc. These keywords define the structure and flow of your C program.

All C language keywords:
`auto, break, case, char, const, continue, default, do, double, else, enum, extern, float, for, goto, if, int, long, register, return, short, signed, sizeof, static, struct, switch, typedef, union, unsigned, void, volatile, while`

Identifiers in C
Identifiers are names you give to various elements in your program, such as variables, functions, arrays, etc. They are user-defined and help you to label and identify different parts of your code. An identifier must start with a letter (uppercase or lowercase) or an underscore `_`, followed by letters, digits, or underscores.

Example:
```
int main() {
    int number = 5; // 'int' is a keyword, 'number' is an identifier
    return 0; // 'return' is a keyword
}
```

data types define the type of data a variable can hold, which helps the compiler manage memory efficiently.

1. **Basic Data Types**
These are the fundamental data types in C:

- **int**:
  - **Description**: Stores Integers numbers (integers).
  - **Size**: Typically 4 bytes.
  - **Range**: -2,147,483,648 to 2,147,483,647.
  - **Example**: `int age = 25;`

- **float**:
  - **Description**: Stores numbers with decimal points (floating-point numbers).
  - **Size**: Typically 4 bytes.
  - **Range**: ~1.2E-38 to 3.4E+38.
  - **Example**: `float temperature = 36.6;`

- **double**:
  - **Description**: Stores decimal numbers with double precision (more accuracy than `float`).
  - **Size**: Typically 8 bytes.
  - **Range**: ~2.2E-308 to 1.7E+308.
  - **Example**: `double distance = 12345.6789;`

- **char**:
  - **Description**: Stores a single character.
  - **Size**: 1 byte.
  - **Range**: -128 to 127 (signed) or 0 to 255 (unsigned).
  - **Example**: `char grade = 'A';`

2. **Derived Data Types**
These are built from the basic data types:

- **Arrays**:
  - **Description**: A collection of elements of the same type stored in a sequence.
  - **Example**: `int scores[5] = {90, 85, 78, 92, 88};`

- **Pointers**:
  - **Description**: Stores the memory address of another variable.
  - **Example**: `int *p = &age;`

- **Structures**:
  - **Description**: A group of different data types under one name.
  - **Example**:
    ```c
    struct Student {
        char name[50];
        int age;
        float marks;
    };
    ```

- **Unions**:
  - **Description**: Similar to structures, but all members share the same memory, meaning only one member can hold a value at a time.
  - **Example**:
    ```c
    union Data {
        int i;
        float f;
        char str[20];
    };
    ```

- **Enumerations (enum)**:
  - **Description**: A set of named integer constants.
  - **Example**:
    ```c
    enum Days {Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday};
    ```

3. **Void Type**

- **void**:
  - **Description**: Represents no data type or "nothing." Commonly used in functions that do not return a value.
  - **Example**:
    ```c
    void printMessage() {
        printf("Hello, World!");
    }
    ```

4. **Modifiers**
Modifiers change the properties of basic data types:

- **signed**:
  - **Description**: Allows storing both positive and negative values.

  - **Example**: `signed int temperature = -10;`

- **unsigned**:
  - **Description**: Stores only positive values, allowing a larger positive range.
  - **Example**: `unsigned int distance = 100;`

- **short**:
  - **Description**: Reduces the size of an `int` to save memory.
  - **Size**: Typically 2 bytes.
  - **Range**: -32,768 to 32,767.
  - **Example**: `short int smallNumber = 32767;`

- **long**:
  - **Description**: Increases the size of an `int` or `double` for storing larger numbers.
  - **Size**: Typically 8 bytes for `long int`.
  - **Range**: -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807.
  - **Example**: `long int largeNumber = 1000000L;

Variables in C

Variables are containers for storing data values. In C, you must declare a variable before you can use it. When you declare a variable, you need to specify the type of data it will hold, such as an integer (`int`), a character (`char`), or a floating-point number (`float`).
- **Declaration**: This is when you tell the compiler about the variable's type and name.
  int age;
- **Initialization**: This is when you assign a value to the variable.
  age = 25;
- **Declaration and Initialization Together**:
  int age = 25;

Rules for Declarations of Variables in C language

1.Name: Variable names can only contain letters, numbers, underscores, and lowercase or uppercase letters.
2.They cannot start with a digit or special characters like #, @, or $.
3.Names also cannot contain whitespaces or hyphens.
4.Reserved keywords like int, char, and float cannot be used as variable names because they have special meanings in the language.
5.Variable names are also case-sensitive.
6.Data type: Variables should be declared with their data type, or the compiler will generate errors. For example, int a = 2 is valid, but a = 2 is not.

Types of Variables

- **Local Variables**: Declared inside a function and can only be used within that function.
  ```c
  void function() {
      int x = 10; // x is a local variable
  }
  ```
- **Global Variables**: Declared outside any function and can be accessed by any function in the program.
  ```c
  int x = 10; // x is a global variable
  void function() {
      printf("%d", x); // can access x
  }
  ```
- **Static Variables**: Retain their value even after the function ends. They are initialized only once.
  ```c
  void function() {
      static int x = 0;
      x++;
      printf("%d", x);
  }
  ```

Constants in C

Constants are values that do not change during the execution of a program. They are like variables, but once assigned a value, it cannot be altered.

Types of Constants

1. **Integer Constants**: All Integers numbers without any decimal point.
   ```c
   const int age = 25;
   ```
2. **Floating-point Constants**: Numbers with a decimal point.
   ```c
   const float pi = 3.14;
   ```
3. **Character Constants**: Single characters enclosed in single quotes.
   ```c
   const char grade = 'A';
   ```
4. **String Constants**: Sequence of characters enclosed in double quotes.
   ```c
   const char name[] = "Asif";
   ```

Defining Constants

- **Using `const` Keyword**: Makes a variable constant, meaning its value cannot be changed.
  ```c
  const int year = 2024;
  ```
- **Using `#define` Directive**: Preprocessor directive used to define constant values. It does not occupy memory space like a variable.
  ```c
  #define PI 3.14159
  ```
  Here, `PI` is a constant that will replace every occurrence of `PI` with `3.14159` during compilation.

Wednesday Assesment

1. write a program to find area of rectangle
2. write a program to find area of square
3. write a program to find area of circle