Keywords in C

Keywords in C are ***reserved words*** that have predefined meanings and are part of the C language syntax. These keywords cannot be used as ***variable names, function names,*** or any other identifiers within the program except for their intended purpose. They are used to define the structure flow and behavior of a C program.

The ***compiler*** recognizes a defined set of keywords in the C programming language. These keywords have ***specialized purposes*** and play critical roles in establishing the logic and behavior of a program. Here are some characteristics of C keywords:

- **Reserved:** The C language ***reserves keywords*** are those keywords that cannot be used as identifiers in programs. Using a ***keyword*** as a variable name or other identifier will cause a ***compilation error***.
- **Predefined Meaning:** Each ***keyword*** has a specific meaning that is assigned by the C language. These meanings are built into the ***C language's grammar*** and ***syntax*** and the compiler interprets them accordingly.
- **Specific Use:** ***Keywords*** are designed for specific purposes and contexts within the C language. They define ***control structures, data types, flow control,*** and other language constructs. Attempting to use a keyword outside of its intended purpose will result in a compilation error.
- **Standardized:** C language ***keywords*** are standardized across different compilers and implementations. It ensures the consistency and portability of C programs across different platforms and environments.

Here is the list of **32** standard keywords in C:

1. auto
2. break
3. case
4. char
5. const
6. continue
7. default
8. do
9. double
10. else
11. enum
12. extern
13. float

14. for
15. goto
16. if
17. inline
18. int
19. long
20. register
21. restrict
22. return
23. short
24. signed
25. sizeof
26. static
27. struct
28. switch
29. typedef
30. union
31. unsigned
32. void

These keywords are reserved and have special meanings in the C language. They cannot be used as identifiers (like variable names, function names, etc.).

Here's a detailed explanation of each **C** keyword:

1. **auto**
    - Used to declare automatic variables. By default, local variables are automatically declared as auto, so it's rarely used explicitly.
    - Example: auto int x = 10;
2. **break**
    - Used to break out of loops (for, while, do-while) or a switch statement.
    - Example:

```
for (int i = 0; i < 10; i++) {
If (i == 5) {
```

```
        break;
      }
    }
```

3. **case**
   - Defines a branch in a switch statement. Each case corresponds to a specific value of the switch expression.
   - Example:

```
switch (n) {
 case 1: printf("One"); break;
 case 2: printf("Two"); break:
   }
```

4. **char**
   - Used to declare a character data type, which is typically a single byte.
   - Example: char letter = 'A';
5. **const**
   - Specifies that the value of a variable is constant and cannot be modified.
   - Example: const int max_value = 100;
6. **continue**
   - Skips the remaining part of the loop and goes back to the loop condition.
   - Example:

```
for (int i = 0; i < 10; i++) {
if (i == 5) {
 continue;
 }
 printf("%d ", i);
}
```

7. **default**
   - Specifies the default case in a switch statement, used if no matching case is found.
   - Example:

```
switch (n) {
 case 1: printf("One"); break;
```

```
default: printf("Unknown");
    }
```

8. **do**
    - Defines the start of a do-while loop, which guarantees the loop runs at least once before checking the condition.
    - Example:

```
int i = 0;
do {
printf("%d", i);
  i++;
} while (i < 5);
```

9. **double**
    - Used to declare a variable of type double precision floating point number (larger range and precision than float).
    - Example: double pi = 3.14159;
10. **else**

- Specifies an alternative block of code to execute if the condition in an if statement is false.
- Example:

```
if (x > 0) {
 printf("Positive");
 } else {
printf("Non-positive");
}
```

11. **enum**

- Defines an enumeration, a user-defined data type that consists of a set of named integer constants.
- Example:

```
enum color {RED, GREEN, BLUE};
```

12. **extern**

- Declares a variable or function that is defined in another file or outside the current scope.
- Example:

```
extern int x; // Declares that x is defined elsewhere
```

13. **float**

- Used to declare a variable of type floating-point (single precision).
- Example: float radius = 5.7;

14. **for**

- Defines the start of a for loop, commonly used for iterating over a range or collection.
- Example:

```
for (int i = 0; i < 10; i++) {
printf("%d ", i);
 }
```

15. **goto**

- Transfers control to a specific labeled statement. Not recommended for use as it makes the code harder to follow.
- Example:

```
if (condition) {
 goto label;
  }
label: printf("Jumped here");
```

16. **if**

- Defines a conditional statement that executes a block of code if the condition is true.
- Example:

```
if (x > 0) {
printf("Positive");
}
```

17. **inline**

- Suggests to the compiler to attempt to inline the function (replace the function call with the function code) to optimize performance.
- Example:

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

18. **int**

- Used to declare a variable of type integer (whole number).

- Example: int age = 25;

19. **long**

- Used to declare a long integer variable (typically 32 or 64 bits, depending on the system).
- Example: long distance = 100000;

20. **register**

- Suggests to the compiler to store a variable in a CPU register for faster access. It is rarely used in modern compilers.
- Example:

```
register int count;
```

21. **restrict**

- Specifies that a pointer is the only reference to a given object, used for optimization purposes.
- Example:

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

22. **return**

- Exits a function and optionally returns a value to the calling function.
- Example:

```
return 0; // returns integer 0 from a function
```

23. **short**

- Declares a short integer (usually 16 bits).
- Example: short x = 100;

24. **signed**

- Specifies that an integer can store both positive and negative values (default for int in most systems).
- Example: signed int number;

25. **sizeof**

- Returns the size (in bytes) of a variable or data type.
- Example: printf("%zu", sizeof(int));

26. **static**

- Declares a variable or function with local scope but retains its value between function calls.
- Example:

```
static int count = 0;
```

27. **struct**

- Defines a structure, a user-defined data type that groups different data types together.
- Example:

```
struct person {
char name[50];
 int age;
 };
```

28. **switch**

- Defines a multi-way branch, allowing control flow based on the value of an expression.
- Example:

```
 switch (x) {
case 1: printf("One"); break;
default: printf("Unknown");
}
```

29. **typedef**

- Creates a new type name (alias) for an existing data type.
- Example:

```
typedef unsigned int uint;
```

30. **union**

- Defines a union, a special data type that allows storing different data types in the same memory location.
- Example:

```
 Union data {
 int i;
float f;
 };
```

31. **unsigned**

- Declares an integer type that can only hold non-negative values (positive numbers and zero).
- Example: unsigned int positiveNumber = 5;

32. **void**

- Used to declare a function that does not return a value, or to specify a pointer with no specific type (void pointer).

- Example:

```
void function() {
// No return value
}
```