Keywords in C Language FAQs

C Programming Journey

Below are some of the most frequently asked questions about the list of keywords in C programming:

1 What is the volatile keyword in C?

The volatile keyword in C is a type qualifier that is used to indicate to the compiler that the value of a variable can change unexpectedly. When a variable is declared as volatile, the compiler assumes that its value can be changed by hardware or software outside of the control of the program, and it will not perform any optimizations that would assume otherwise.

The volatile keyword is commonly used in embedded systems programming or in programs that interact with hardware devices. In these cases, the value of a variable may be changed by an external process, such as an interrupt service routine, without the program's knowledge.

By declaring a variable as volatile, the compiler will always read its value from memory rather than using a cached value, ensuring that the most up-to-date value is used in the program.

Here's an example of how to declare a volatile variable:

volatile int value;

In this example, the variable value is declared as a volatile int. This means that the value of value can change unexpectedly, and the compiler will not make any assumptions about its value.

Here's an example where the `volatile` keyword is used in C:

```c
#include <stdio.h>

int main() {
    volatile int sensorValue = 0;

    while (1) {
        // Read sensor value from hardware
        // In a real scenario, this value might change unexpectedly
        sensorValue = /* Read from hardware */ ;

        // Print sensor value
        printf("Sensor Value: %d\n", sensorValue);
    }

    return 0;
}
```

Explanation:

1. **Include Header**: The `#include <stdio.h>` directive includes the standard input-output header file for functions like `printf()`.
2. **Declare Variable**: `volatile int sensorValue = 0;` declares an integer variable `sensorValue` and marks it as `volatile`. This means that the value of `sensorValue` may change unexpectedly, perhaps due to external hardware or asynchronous operations.
3. **Main Function**: The `main()` function is the entry point of the program.
4. **Infinite Loop**: `while (1)` creates an infinite loop, which continuously reads and prints the sensor value.
5. **Read Sensor Value**: Inside the loop, `sensorValue` is updated with the current sensor reading. In a real scenario, this value might come from hardware and could change at any time.

6. **Print Sensor Value**: The current value of `sensorValue` is printed using `printf()`.

By marking `sensorValue` as `volatile`, we inform the compiler that its value may change unexpectedly, so it should not optimize away accesses to it or assume that its value remains constant. This ensures that the compiler always reads the latest value of `sensorValue` from memory during each access, even if it appears redundant from the compiler's perspective.

2 What is an extern keyword in C?

The extern keyword in C is used to declare a variable or function that is defined in another source file. When a variable or function is declared with the extern keyword, it indicates to the compiler that the actual definition of the variable or function can be found in another source file and that the declaration is simply an external reference to it.

The extern keyword is often used in larger projects to manage the scope of variables and functions across multiple source files.

For example, if you have a variable or function that you need to use in multiple source files, you can declare it with the extern keyword in each source file that needs to access it, and then define it in a single source file.

Here's an example of how to declare an extern variable:

```
// file1.c
extern int my_variable;
// file2.c
#include
int my_variable = 10;
int main()
{
   printf("%d\n", my_variable);
   return 0;
}
```

In this example, the variable my_variable is declared with the extern keyword in file1.c. The definition of my_variable is then provided in file2.c. When the program is compiled and linked, the definition of my_variable from file2.c will be linked to the declaration in file1.c, and both source files will be able to access the same variable.

3 What is the register keyword in C?

The register keyword in C is a type qualifier that is used to suggest to the compiler that a variable should be stored in a register instead of in memory. When a variable is declared as a register, it indicates to the compiler that the variable will be heavily used and that it is a good candidate to be stored in a register.

Registers are a small amount of fast storage that is built into the processor. By storing a frequently used variable in a register, the program can access it more quickly, as the processor can access the value stored in a register much faster than it can access a value stored in memory.

It's important to note that the compiler is free to ignore the register keyword and store the variable in memory if it determines that it is not beneficial to store it in a register. The use of the register keyword is simply a suggestion to the compiler.

Here's an example of how to declare a register variable:

register int value;

In this example, the variable value is declared as a register int. This means that the compiler is being asked to store the variable in a register if possible, as it is expected to be heavily used in the program.

multiple-choice questions:

1. A whole number with a decimal point is known as:

    - a. floating point number.
    - b. character.
    - c. integer.
    - d. none.

    a. floating point number. A floating-point number is a positive or negative whole number with a decimal point. It allows for representing real numbers that may have fractional components.

2. The declaration `unsigned u` indicates:

    - a. u is a character.
    - b. u is an unsigned integer.

- c. u is unsigned character.
- d. u is unsigned long integer.

3. Which statement must not end with a semicolon:

- a. `#define`
- b. variable declaration
- c. assignment
- d. none

4. Point out the valid variable names:

- a. gross salary
- b. gross-salary
- c. AVG
- d. AVG.

5. If `a` is an integer variable, `a = 5 / 2` will return a value:

- a. 2.5
- b. 3
- c. 2
- d. 0

6. The expression `a = 7 / 22 * (3.14 + 2) * 3 / 5` is evaluated to:

- a. 8.28
- b. 6.28
- c. 3.14
- d. 0

7. Hierarchy decides which operator:

- a. is most important
- b. is used first
- c. is fastest
- d. operates on largest numbers

8. The expression `a = 4 + 2 % 8` evaluates to:

- a. -6
- b. 6

- c. 4
- d. none of the above

9. In C, a variable cannot contain:

   - a. blank spaces
   - b. hyphen (-)
   - c. semicolon
   - d. none

10. Arithmetic instructions cannot contain:

    - a. variables
    - b. constants
    - c. variable names on the right of equal
    - d. variable names on the left of equal

11. Which of the following is the odd one out:

    - a. +
    - b. -
    - c. /
    - d. **

12. What will be the value of `d` assuming it to be float after the operation `d = 2 / 7.0`:

    - a. 0
    - b. 0.2857
    - c. cannot be determined
    - d. none of the above

13. The expression `a = 30 * 1000 + 2768` evaluates to:

    - a. 32768
    - b. -32768
    - c. 113040
    - d. 0

Answers: 1(a) 2(b) 3(a) 4(c) 5(c) 6(d) 7(b) 8(b) 9(b) 10(d) 11(c) 12(d) 13(b)

Theory Questions

1. Explain what are data types? Name all the data types in c language with their size in bytes (in dos operating system). List all the format specifiers. Also write the formula by which we can calculate the range of a given data type.
2. Calculate the range of int by this.
3. How many keywords are present in C. Give the name of all the keywords?
4. Explain all the naming rules of a variable in c.
5. Characteristics of a program.
6. Name all the escape sequences.
7. What do you mean by precedence of operators? List all the available
8. What do you mean by typecasting? Explain implicit and explicit typecasting
9. operators in c in the order of their precedence. (higher to lower)

Answers to the Questions:

1. **Data Types in C**:
    - Data types in C specify the type of data that a variable can hold. Here are the data types in C with their sizes in bytes (in DOS operating system):
        - `char`: 1 byte
        - `int`: 2 bytes
        - `float`: 4 bytes
        - `double`: 8 bytes
    - Format specifiers:
        - `%c` for `char`
        - `%d` for `int`
        - `%f` for `float`
        - `%lf` for `double`
    - Formula for calculating the range of a given data type: For signed integers, it's `-2^(N-1)` to `2^(N-1) - 1`, and for unsigned integers, it's `0` to `2^N - 1`, where `N` is the number of bits used to represent the data type [2].
2. **Range of `int`**:
    - For a signed integer in DOS operating system, it ranges from `-32768` to `32767`.
3. **Keywords in C**:

- There are 32 keywords in C language. Some of them include `int`, `char`, `float`, `if`, `else`, `while`, `for`, `return`, `switch`, and `void` [3].

4. **Naming Rules of Variables in C**:

   - Variable names in C must begin with a letter (a-z, A-Z) or underscore (_).
   - They can be followed by letters, digits (0-9), or underscores.
   - Variable names are case-sensitive.
   - Reserved words (keywords) cannot be used as variable names.

5. **Characteristics of a Program**:

   - Correctness, readability, maintainability, efficiency, modularity, and reliability are essential characteristics of a program.

6. **Escape Sequences**:

   - Some common escape sequences in C include `\n` for newline, `\t` for tab, `\b` for backspace, `\r` for carriage return, `\\` for a backslash, `\"` for a double quote, and `\'` for a single quote.

7. **Precedence of Operators**:

   - Precedence of operators determines the order of operations in an expression. Operators with higher precedence are evaluated first. In C, operators with higher precedence are:
     - `()`, `[]`, `->`, `.` (parentheses, brackets, arrow, dot)
     - `!`, `~`, `++`, `--` (unary operators)
     - `*`, `/`, `%` (multiplication, division, modulus)
     - `+`, `-` (addition, subtraction)
     - `<<`, `>>` (bitwise shift)
     - `==`, `!=`, `<`, `<=`, `>`, `>=` (comparison)
     - `&` (bitwise AND)
     - `^` (bitwise XOR)
     - `|` (bitwise OR)
     - `&&` (logical AND)
     - `||` (logical OR)
     - `?:` (conditional)
     - `=` (assignment)

8. **Typecasting**:

- Typecasting in C is the process of converting a variable from one data type to another. It can be implicit or explicit.
- **Implicit Typecasting**: Also known as automatic type conversion, it's done by the compiler automatically when compatible data types are involved in an expression.
- **Explicit Typecasting**: Also known as manual type conversion, it's done by the programmer using casting operators to convert data from one type to another.

9. **Operators in C in the Order of Their Precedence (Higher to Lower)**

    1. **Postfix Increment (++) and Decrement (--)**
    2. **Prefix Increment (++) and Decrement (--)**
    3. **Unary Plus (+) and Minus (-)**
    4. **Logical NOT (!)**
    5. **Bitwise NOT (~)**
    6. **Multiplication (*) and Division (/)**
    7. **Modulus (%)**
    8. **Addition (+) and Subtraction (-)**
    9. **Bitwise Left Shift (<<) and Right Shift (>>)**
    10. **Relational Operators (<, <=, >, >=)**
    11. **Equality Operators (==, !=)**
    12. **Bitwise AND (&)**
    13. **Bitwise XOR (^)**
    14. **Bitwise OR (|)**
    15. **Logical AND (&&)**
    16. **Logical OR (||)**
    17. **Conditional Operator (?:)**
    18. **Assignment Operators (=, +=, -=, =, /=, %=, <<=, >>=, &=, ^=, |=)*
    19. **Comma Operator (,)**

Sum First Last Digits

Lab Ex1:

If a four digit number is input through the keyboard, write a program to obtain the sum of the first and last digit of the number.

Below is the algorithm for a C program that obtains the sum of the first and last digits of a four-digit number input through the keyboard:

1. Start the program.
2. Declare variables for storing the four-digit number, the first digit, the last digit, and the sum.
3. Prompt the user to enter a four-digit number.
4. Read the input four-digit number from the user.
5. Extract the first digit of the number by dividing it by 1000.
6. Extract the last digit of the number by taking the remainder of the number divided by 10.
7. Calculate the sum of the first and last digits.
8. Display the sum to the user.
9. End the program.

Here's a flowchart representing the algorithm:

```
Start
|
V
Declare variables num, first_digit, last_digit, sum
|
V
Prompt user to enter a four-digit number
|
V
Read input into variable num
|
V
first_digit = num / 1000
|
V
last_digit = num % 10
|
V
sum = first_digit + last_digit
|
V
Display sum to the user
|
V
End
```

Lab Ex1 C Coce for the Algorithm :

```c
#include <stdio.h>

int main() {
    int num, firstDigit, lastDigit, sum;

    // Input a four-digit number
    printf("Enter a four-digit number: ");
    scanf("%d", &num);

    // Extract the first digit by dividing by 1000
    firstDigit = num / 1000;

    // Extract the last digit by taking the remainder when divided by 10
    lastDigit = num % 10;

    // Calculate the sum of the first and last digits
    sum = firstDigit + lastDigit;

    // Output the sum
    printf("Sum of the first and last digit is %d\n", sum);

    return 0;
}
```

number range printer

Lab Ex2:

Print the range of a number. E.g. number 78 is between 70 and 79, 102 is between 100 and 109.

Algorithm for a C Program to Print the Range of a Number: This algorithm takes an input number, calculates the range by finding the nearest multiple of 10 below the input number as the lower bound, and adding 9 to it for the upper bound, effectively creating a range of 10 numbers centered around the input number.

1. Start the program.
2. Declare variables `num`, `lowerBound`, and `upperBound` of type integer.
3. Prompt the user to enter a number and store it in the variable `num`.
4. Calculate the lower bound of the range:
    - Divide the number by 10 and store the result in `lowerBound`.
    - Multiply `lowerBound` by 10 (Calculate the lower bound of the range by dividing the input number by 10 and then multiplying the result by 10.)
5. Calculate the upper bound of the range:

    - Assign `upperBound` as `lowerBound + 9`.
6. Print the range of the number as `lowerBound` to `upperBound`.
7. End the program.

Lab Ex2 C Coce for the Algorithm:

```c
#include <stdio.h>

int main(void){
    int num, lowerBound, upperBound;
    printf("Enter a number: ");
    scanf("%d", &num);

    //  calculate lower bound of range:
    lowerBound = num / 10;
    lowerBound = lowerBound * 10;

    //  calculate upper bound of range:
    upperBound = lowerBound + 9;

    //  Print the range of the number as `lowerBound` to `upperBound`
    printf("The range of number %d is %d and %d", num, lowerBound, upperBound);

    return 0;
}
```

Another way to calculate the range of a number:

This algorithm calculates the range of a number by finding the nearest multiple of 10 below the input number as the lower bound and adding 10 to it for the upper bound, effectively creating a range of 10 numbers centered around the input number.

1. Start the main function.
2. Declare integer variables `num`, `lowerBound`, and `upperBound`.
3. Prompt the user to input a number.
4. Read the input number and store it in the variable `num`.
5. Calculate the lower bound of the range by subtracting the remainder of `num` divided by 10 from `num` itself.
6. Calculate the upper bound of the range by adding 10 to the lower bound.
7. Output the range along with the input number.
8. End the main function.

C Coce for the Algorithm:

```c
#include <stdio.h>

// C program that prints the range of a number

int main() {
    int num, lowerBound, upperBound;

    // Input a number
    printf("Enter a number: ");
    scanf("%d", &num);

    // Calculate the lower bound of the range
    lowerBound = num - (num % 10);


    // Calculate the upper range
    upperBound = lowerBound + 10;


    // Output the range
    printf("The range of %d is %d to %d\n", num, lowerBound, upperBound);


    return 0;
}
```

Denomination Calculator

Lab Ex3: Print the various denominations of a given rupee. E.g. if a person has 1779 in his pocket the program should print the following. 500 x 3 = 1500, 100 x 2 = 200, 50 x 1 = 50, 20 x 1 = 20, 10 x 0 = 0, 5 x 1 = 5, 2 x 2 = 4, 1 x 0 = 0

This code takes an input amount in rupees, calculates the number of each denomination of rupee notes required to represent the given amount, and prints them along with their respective subtotals. It iterates through each denomination, calculates the number of notes using integer division, updates the amount, and prints the denomination, number of notes, and subtotal.

This algorithm calculates and prints the number of each denomination of rupee notes required to represent the given amount, along with their respective subtotals.

1. Start the main function.
2. Declare integer variables `amount` and denominations of various rupee notes (e.g., 500, 100, 50, 20, 10, 5, 2, 1).
3. Prompt the user to input the amount in rupees.

4. Read the input amount and store it in the variable `amount`.
5. Iterate through each denomination:
    - Calculate the number of notes of the current denomination using integer division.
    - Update the amount by subtracting the product of the number of notes and denomination.
    - Print the denomination, the number of notes, and the subtotal (number of notes * denomination).
6. End the iteration.
7. End the main function.

Lab Ex3 C Coce for the Algorithm:

```c
#include <stdio.h>


int main() {
    // Declare variables
    int amount, denomination;

    // Array to hold various rupee denominations
    int denominations[] = {500, 100, 50, 20, 10, 5, 2, 1};

    // Prompt user to input the amount in rupees
    printf("Enter the amount in rupees: ");

    // Read the input amount
    scanf("%d", &amount);

    // Iterate through each denomination
    for (int i = 0; i < sizeof(denominations) / sizeof(denominations[0]); i++) {
        // Calculate number of notes of the current denomination
        int notes = amount / denominations[i];

        // Update the amount by subtracting the product of notes and denomination
        amount -= notes * denominations[i];

        // Print denomination, number of notes, and subtotal
        printf("%d x %d = %d\n", denominations[i], notes, notes *
denominations[i]);
    }

    return 0;
}
```

Three-Digit Number

Lab Ex4:

WAP to calculate the sum of digits of a three-digit number e.g. 125 is 8.

This program reads a three-digit number from the user, extracts its digits, calculates their sum, and prints the result.

Algorithm:

1. Declare variables to store the three-digit number and the sum of its digits.
2. Prompt the user to input the three-digit number.
3. Read the input number from the user.
4. Extract each digit of the number using the modulus operator (%) and integer division (/).
5. Add each extracted digit to the sum variable.
6. Print the sum of the digits.

Lab Ex4 C Coce for the Algorithm:

```c
#include <stdio.h>

int main() {
    int num, digit, sum = 0;

    // Input the three-digit number
    printf("Enter a three-digit number: ");
    scanf("%d", &num);

    // Extract and sum the digits
    digit = num % 10; // Extract the units digit
    sum += digit;
    num /= 10; // Remove the units digit

    digit = num % 10; // Extract the tens digit
    sum += digit;
    num /= 10; // Remove the tens digit

    digit = num % 10; // Extract the hundreds digit
    sum += digit;

    // Print the sum of the digits
    printf("Sum of digits: %d\n", sum);

    return 0;
}
```

Merge Three Numbers

Lab Ex5:

WAP to merge three numbers. E.g. a= 1, b= 2, c = 8 is 128. Algorithm:

Input: Declare three integer variables a, b, and c.

Read Input: Prompt the user to enter the first number and store it in variable a. Repeat this process for the second and third numbers, storing them in variables b and c respectively.

Merge Numbers: Multiply the first number (a) by 100, the second number (b) by 10, and add the third number (c). Store the result in a variable named merged_number.

Output: Print the merged number using the printf function along with a descriptive message.

Lab Ex5 C Coce for the Algorithm:

```c
#include <stdio.h>

int main() {
    int a, b, c;

    // Read three numbers from the user
    printf("Enter the first number: ");
    scanf("%d", &a);
    printf("Enter the second number: ");
    scanf("%d", &b);
    printf("Enter the third number: ");
    scanf("%d", &c);

    // Merge the three numbers
    int merged_number = a * 100 + b * 10 + c;

    // Print the merged number
    printf("Merged number: %d\n", merged_number);

    return 0;
}
```

Reverse of a Three-Digit Number

Lab Ex6: WAP to print the reverse of a 3-digit number. Algorithm:

- Start
- Read a three-digit number from the user and store it in a variable.
- Extract the digits by dividing the number by 10 and taking the remainder.
- Reconstruct the number in reverse order.
- Print the reversed number.
- End **Lab Ex6 C Coce for the Algorithm:**

```c
#include <stdio.h>

int main() {
    int number, originalNumber, remainder, reversedNumber = 0;

    // Read a three-digit number from the user
    printf("Enter a three-digit number: ");
    scanf("%d", &number);

    originalNumber = number;

    // Extract the digits, reverse, and reconstruct the number
    while (originalNumber != 0) {
        remainder = originalNumber % 10;
        reversedNumber = reversedNumber * 10 + remainder;
        originalNumber /= 10;
    }

    // Print the reversed number
    printf("Reversed number: %d\n", reversedNumber);

    return 0;
}
```

Compute the Gross Salary of Mr. HARISH

Lab Ex7:

Wap Compute the gross salary of Mr. HARISH. Input his basic salary. His DA is 40% of the basic salary, and HRA is 20% of the basic salary.

Algorithm:

- Start

- Read the basic salary of Mr. HARISH from the user and store it.
- Calculate DA (40% of the basic salary) and HRA (20% of the basic salary).
- Add the basic salary, DA, and HRA to get the gross salary.
- Print the gross salary.
- End **Lab Ex7 C Coce for the Algorithm:** To compute Mr. Harish's gross salary, follow these steps:

1. **Input**: Obtain Mr. Harish's basic salary.
2. **Calculate DA**: Determine the Dearness Allowance (DA) as 40% of the basic salary.
3. **Calculate HRA**: Determine the House Rent Allowance (HRA) as 20% of the basic salary.
4. **Compute Gross Salary**: Add the basic salary, DA, and HRA together to get the gross salary.

Here's the code implementing the above algorithm:

```c
#include <stdio.h>

int main() {
    float basicSalary, da, hra, grossSalary;

    // Input basic salary
    printf("Enter Mr. Harish's basic salary: ");
    scanf("%f", &basicSalary);

    // Calculate DA (40% of basic salary)
    da = 0.4 * basicSalary;

    // Calculate HRA (20% of basic salary)
    hra = 0.2 * basicSalary;

    // Compute gross salary
    grossSalary = basicSalary + da + hra;

    // Output the gross salary
    printf("Mr. Harish's gross salary is: %.2f\n", grossSalary);

    return 0;
}
```

This code takes Mr. Harish's basic salary as input, calculates the DA and HRA based on the given percentages, and then computes the gross salary by adding the basic salary, DA, and HRA together.

C Programming Journey