C Programming Assignments & Interview Questions

Variables & Storage Classes

Assignment 1: Variables

1. **Basic Variable Declaration:**

- Declare variables of type `int`, `float`, and `char`.

- Assign values and print them using `printf()`.

Ans :variable declaration : Data type variable name ;(data type maybe int, float, char etc)

```
 #include <stdio.h>
void  main()
{
   // Variable declarations and assignments
   int myint = 25;
   float myfloat = 3.14;
   char mychar = 'A';
   // Printing the values
   printf("Integer value: %d\n", myint);
   printf("Float value: %.2f\n", myfloat);
   printf("Character value: %c\n", mychar);
}
```

2. **User Input and Output:**

- Write a program that asks the user to enter an integer, a floating-point number, and a character, then displays them.

Ans :

```
#include <stdio.h>
void main()
{
   int myint;
   float myfloat;
   char mychar;
   // Asking the user for input
   printf("Enter an integer: ");
   scanf("%d", &myint);
```

```
    printf("Enter a floating-point number: ");

    scanf("%f", &myfloat);

    // Clear the input buffer before reading a character

    while ((getchar()) != '\n'); // Flush newline character left by previous input

    printf("Enter a character: ");

    scanf("%c", &mychar);

    // Displaying the values

    printf("\nYou entered:\n");

    printf("Integer: %d\n", myint);

    printf("Float: %.2f\n", myfloat);

    printf("Character: %c\n", mychar);

}
```

3. **Scope of Variables:**

- Write a program to demonstrate **local and global variables by declaring a global variable and modifying it inside a function.**

Ans : #include <stdio.h>

```
// Global variable

int globalVar = 10;

// Function to modify global variable

void modifyGlobal() {

    // Local variable

    int localVar = 5;

    printf("Inside modifyGlobal function:\n");

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

    printf("Global variable before modification: %d\n", globalVar);

    // Modifying global variable

    globalVar += 20;

    printf("Global variable after modification: %d\n", globalVar);

}

int main() {
    printf("Inside main function:\n");

    printf("Global variable before function call: %d\n", globalVar);

    // Call the function to modify global variable

    modifyGlobal();

    printf("Global variable after function call: %d\n", globalVar);

    // Uncommenting the next line would cause an error, because localVar is not accessible here

    // printf("Local variable: %d\n", localVar);

    return 0;

}
```

Assignment 2:Storage Classes

1.Demonstrating Different Storage Classes:****

- Write a program to show the difference between `auto`, `static`, `extern`, and `register` storage classes.

Ans : #include <stdio.h>

```
// Global variable for extern example

int externVar = 50;

// Function prototype

void demonstrateExtern();

void demonstrateStorageClasses() {

  // ----------- AUTO (default for local variables) -----------

  auto int a = 10;  // 'auto' is optional in modern C

  printf("Auto variable (a): %d\n", a);

  // ----------- REGISTER -----------

  register int r = 20;

  printf("Register variable (r): %d\n", r);

  // Note: You can't use '&r' here, because register variables can't have their address taken.

  // ----------- STATIC -----------

  static int s = 0;

  s++;

  printf("Static variable (s): %d\n", s);

}
```

```
int main() {
   printf("----- Call 1 -----\n");
   demonstrateStorageClasses();
   printf("----- Call 2 -----\n");
   demonstrateStorageClasses();
   printf("----- Call 3 -----\n");
   demonstrateStorageClasses();
   // ----------- EXTERN -----------
   printf("Extern variable (externVar): %d\n", externVar);
   demonstrateExtern();
   return 0;
}
// Function to demonstrate extern access
void demonstrateExtern() {
   extern int externVar;  // Declare extern variable (already defined above)
   externVar += 10;
   printf("Extern variable modified in function: %d\n", externVar);
}
```

2. **Static vs Non-Static Variables:**

- Write a function containing a static variable and call it multiple times to observe how it retains its value.

Ans : #include <stdio.h>

```
// Function with a static and a non-static variable
void counterFunction() {
   int nonStaticVar = 0;       // Reinitialized every time
   static int staticVar = 0;   // Initialized only once and retains value
   nonStaticVar++;
   staticVar++;
   printf("Non-static variable: %d\n", nonStaticVar);
   printf("Static variable: %d\n", staticVar);
}
```

```
int main() {
   printf("---- Call 1 ----\n");
   counterFunction();
   printf("---- Call 2 ----\n");
   counterFunction();
   printf("---- Call 3 ----\n");
   counterFunction();
   return 0;
}
```

3. **Extern Variable Usage:**

 - Declare an ` extern` variable in one file and modify it in another file.

Ans : #include <stdio.h>

```
int sharedVar = 10;// Define the global variable in file1.c
// Function declaration
void modifyExtern();
int main() {
   printf("Initial value of sharedVar: %d\n", sharedVar);
   // Call function from file2.c
   modifyExtern();
   printf("Value of sharedVar after modification: %d\n", sharedVar);
   return 0;
}
// Access the global variable declared in file1.c
extern int sharedVar;//
void modifyExtern()
{
   sharedVar += 5;
   printf("sharedVar modified inside file2.c: %d\n", sharedVar);
}
```

Interview Questions: Variables & Storage Classes

1. **Understanding register Storage Class:**

- Write a program that declares an integer variable using the register storage class.

- Attempt to get its address using the & operator. Explain why this behavior occurs.

```
Ans : #include <stdio.h>

int main() {

   register int regVar = 10;

   printf("Value of register variable: %d\n", regVar);

   // Uncommenting the following (printf) line will cause a **compilation error**

   // because we can't get the address of a register variable.

   //printf("Address of register variable: %p\n", &regVar);

   return 0;

}
```

register tells the compiler to **store the variable in a CPU register** instead of RAM, if possible.

CPU registers **don't have memory addresses accessible in user code**, unlike regular variables in RAM.

That's why trying to use the address-of operator (&regVar) causes a **compilation error**.

2. **Global vs Local Variables:**

- Declare a global variable and modify it inside `main()`.

- Declare a local variable inside a block and print its value both inside and outside the block.

- Explain what happens when trying to access the local variable outside its block.

Ans : global variable means variable are declare outside the block

Local variable means variable declare inside a block of code

```
#include <stdio.h>

int globalCounter = 0;//global variable declaration

int main() {

   printf("Initial value of globalCounter: %d\n", globalCounter);

   globalCounter = 10;//modified global variable

   printf("Modified value of globalCounter: %d\n", globalCounter);

   return 0; }
```

```
#include <stdio.h>

int main() {

{

    int localVar = 42;  // Local variable inside this block

     printf("Inside the block: localVar = %d\n", localVar);

  }

  // Trying to access localVar outside the block will cause an error so Uncomment the next line
//to see what happens

  // printf("Outside the block: localVar = %d\n", localVar);

  return 0;

}
```

When we try to access a **local variable** outside of the block in which it was declared, we'll encounter a **compilation error**. This happens because **local variables have block-level scope**, meaning they only exist and are accessible within the block (or function) in which they are declared.

Local variables are temporary and are destroyed once the program exits the block or function in which they were declared. They **cannot** be accessed outside the block or function .

3. **Scope of Static Variables:**

- Write a program where a **static variable is declared inside a loop.**

- Print its value on each iteration and compare its behavior with a non-static variable declared inside the same loop

Ans : A static variable **retains its value between function calls**, but its **scope is limited** to the block or file in which it is declared.

```
#include <stdio.h>

void main() {

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

     int normalVar = 0;        // Non-static: reinitialized every iteration

     static int staticVar = 0;   // Static: initialized once and retains its value inside the block

     normalVar++;

     staticVar++;

     printf("Iteration %d:\n", i + 1);

     printf("  Non-static variable: %d\n", normalVar);
```

```
        printf("  Static variable:    %d\n", staticVar);
    }
    return 0;
}
```

Assignment 3: Operators

1. **Arithmetic Operators:**

- Take two numbers as input and perform addition, subtraction, multiplication, division, and modulus operations.

Ans : #include <stdio.h>

```
void main()
{
    int num1, num2;
    // Taking input from the user
    printf("Enter the first number: ");
    scanf("%d", &num1);
    printf("Enter the second number: ");
    scanf("%d", &num2);
    // Performing arithmetic operations
    printf("Addition: %d + %d = %d\n", num1, num2, num1 + num2);
    printf("Subtraction: %d - %d = %d\n", num1, num2, num1 - num2);
    printf("Multiplication: %d * %d = %d\n", num1, num2, num1 * num2);
    if (num2 != 0)//check the number division by zero
    {
        printf("Division: %d / %d = %d\n", num1, num2, num1 / num2);
        printf("Modulus: %d %% %d = %d\n", num1, num2, num1 % num2);
    } else {
        printf("Division and Modulus by zero are not allowed.\n");
    }
}
}
```

2. **Relational and Logical Operators:**

- Write a program that takes two integers as input and:

- Compares them using `>`, `=`, `<=`, `==`, `!=`.

- Uses logical operators (`&&`, `||`, `!`) to check multiple conditions.

Ans :
```
#include <stdio.h>
void main()
{
   int a, b;//input two integers
   printf("Enter first integer: ");
   scanf("%d", &a);
   printf("Enter second integer: ");
   scanf("%d", &b);
   // Relational operators
   printf("%d > %d  : %d\n", a, b, a > b);
   printf("%d < %d  : %d\n", a, b, a < b);
   printf("%d >= %d : %d\n", a, b, a >= b);
   printf("%d <= %d : %d\n", a, b, a <= b);
   printf("%d == %d : %d\n", a, b, a == b);
   printf("%d != %d : %d\n", a, b, a != b);
   // Logical operators
   printf("(%d > 0 && %d > 0)  : %d\n", a, b, (a > 0 && b > 0));
   printf("(%d > 0 || %d > 0)  : %d\n", a, b, (a > 0 || b > 0));
   printf("!(%d == %d)        : %d\n", a, b, !(a == b));
}
```

3. **Assignment Operators:**

- Write a program that demonstrates the use of assignment operators (`=`, `+=`, `-=`, `*=`, `/=`, `%=`)

Ans :
```
#include <stdio.h>
void main()
{
   int a;
```

```
    a = 10;//simple assignment
    printf("Initial value of a: %d\n", a);
    a += 5;  // a = a + 5  Addition assignment
    printf("After a += 5: %d\n", a);
    a -= 3;  // a = a - 3Subtraction assignment
    printf("After a -= 3: %d\n", a);
    a *= 2;  // a = a * 2 Multiplication assignment
    printf("After a *= 2: %d\n", a);
    a /= 4;  // a = a / 4 Division assignment
    printf("After a /= 4: %d\n", a);
    a %= 3;  // a = a % 3 Modulus assignment
    printf("After a %%= 3: %d\n", a);
}
```

4. **Increment and Decrement Operators:**

- Take an integer input and demonstrate:

- **Pre-increment (`++x`) vs. Post-increment (`x++`).**

- **Pre-decrement (`--x`) vs. Post-decrement (`x--`).**

Ans :
```
#include <stdio.h>
int main() {
    int x;//integer input from user
    printf("Enter an integer: ");
    scanf("%d", &x);
    printf("\n--- Increment and Decrement Demonstration ---\n");
    // Pre-increment
    printf("Original x: %d\n", x);
    printf("Pre-increment (++x): %d\n", ++x);  // Increment first, then use the value
    printf("After pre-increment, x: %d\n", x);
    // Post-increment
    printf("\nPost-increment (x++): %d\n", x++);  // Use the value, then increment
    printf("After post-increment, x: %d\n", x);
    // Pre-decrement
```

```
   printf("\nPre-decrement (--x): %d\n", --x);  // Decrement first, then use

   printf("After pre-decrement, x: %d\n", x);

   // Post-decrement

   printf("\nPost-decrement (x--): %d\n", x--);  // Use the value, then decrement

   printf("After post-decrement, x: %d\n", x);

   return 0;

}
```

5. **Ternary Operator:**

- Write a program that takes two numbers as input and prints the larger one using a **ternary operator**.

```
Ans : #include <stdio.h>

void main() {

   int num1, num2, max;   //input two numbers

   printf("Enter the first number: ");

   scanf("%d", &num1);

   printf("Enter the second number: ");

   scanf("%d", &num2);

   // Using ternary operator to find the larger number

   max = (num1 > num2) ? num1 : num2;

   printf("\nThe larger number is: %d\n", max);

}
```

Interview Questions: Operators

1. **Even or Odd Using Logical AND (`&&`)**

 - Write a program that checks whether an integer is even or odd **without using the modulus (`%`) operator**, but instead using the **logical AND (`&&`) operator**.

```
Ans : #include <stdio.h>

void main()

{

   int num;

   printf("Enter an integer: ");

   scanf("%d", &num);
   if ((num / 2) * 2 == num && 1)// Check even using division and logical AND (no % or & used)
 {
      printf("%d is Even.\n", num);
   } else {
      printf("%d is Odd.\n", num);
   }
}
```

2. **Chained Assignment Operators:**

 - Write a program that initializes a variable and assigns values to multiple variables using a **chained assignment (` a = b = c = 10;`).**

- Modify ` c` and print all values to observe the effect on ` a` and ` b` .

Ans : #include <stdio.h>

```
void main() {
   int a, b, c;
   // Chained assignment: All variables get the value 10
   a = b = c = 10;//chained assignment all variables are assigned to the value 10
   printf("Initial values after chained assignment:\n");
   printf("a = %d\n", a);
   printf("b = %d\n", b);
   printf("c = %d\n", c);
   c = 25;//modifying the only  variable c
   printf("\nAfter modifying c:\n");
   printf("a = %d\n", a);
   printf("b = %d\n", b);
   printf("c = %d\n", c);
}
```

3. **Comparing Floating-Point Numbers:**

 - Write a program that takes **two floating-point numbers as input and checks whether they are **equal** using a **relational operator**.**

 - Handle floating-point precision errors correctly.

Ans : #include <stdio.h>

```
#include <math.h>  // For fabs() means give absolute value of floating point output
```

```
void main() {
   float num1, num2;
   const float epsilon = 0.0001;  // Acceptable error margin
   // Input two float numbers
   printf("Enter the first floating-point number: ");
   scanf("%f", &num1);
   printf("Enter the second floating-point number: ");
   scanf("%f", &num2);
   // Check if the numbers are equal within epsilon
   if (fabs(num1 - num2) < epsilon) {
      printf("The numbers are considered equal.\n");
   } else {
      printf("The numbers are NOT equal.\n");
   }
}
```

4. **Swapping Two Variables Without Using Bitwise Operators:**

- Swap two variables **without using a third variable and **without using bitwise XOR (`^`)**. - Avoid the standard `+` and `-` method—use a different approach.**

Ans : #include <stdio.h>

```
int main()
{
   int a, b;
   printf("Enter two integers:\n");
   scanf("%d %d", &a, &b);
   printf("\nBefore swapping: a = %d, b = %d\n", a, b);
   // Swap using multiplication and division
   if (a != 0 && b != 0) {  // Prevent division by zero
      a = a * b;
      b = a / b;
      a = a / b;
```

```
        printf("After swapping: a = %d, b = %d\n", a, b);
    } else {
        printf("Swapping using multiplication/division won't work with 0.\n");
    }
    return 0;
}
```

5. **Largest of Three Numbers Without Using `if-else` or Ternary Operator:**

- Write a program that takes three numbers as input and prints the largest one **without using `if-else` or the ternary (`? :`) operator**.

Ans : #include <stdio.h>

```
#include <math.h>  // For fmax()
int main() {
    double a, b, c, max;
    printf("Enter three numbers:\n");
    scanf("%lf %lf %lf", &a, &b, &c);
    // Find the maximum using fmax
    max = fmax(a, fmax(b, c));
    printf("The largest number is: %.2lf\n", max);
    return 0;
}
```

6. **Complex Expression Evaluation:**

- Evaluate the following expression in a program and explain the output: int a = 5, b = 10, c = 15; int result = a++ + --b * c / 5; printf("%d\n", result);

- Break down the order of execution step by step

Ans : the given program is

```
#include <stdio.h>
int main() {
    int a = 5, b = 10, c = 15;
    int result = a++ + --b * c / 5;
    printf("%d\n", result);
    return 0;
}
```

We have to breakdown this program by using order of precedence

Given b=10 it is undergo the pre decrement process –b then become the b=9

Given a=5 it is undergo the post increment process ++a then become the a=6

Result =a++ + --b*c/5

6+9*15/5 → 6+135/5→6+27→ 32// first will multiplication next division then after addition

** Challenge Questions**

1. **Swap Two Numbers Without Using a Third Variable** (using arithmetic operations)**

```
Ans :#include <stdio.h>
int main() {
  int a, b;
  printf("Enter two integers:\n");//input from user
  scanf("%d %d", &a, &b);
  printf("\nBefore swapping: a = %d, b = %d\n", a, b);
  // Swap using addition and subtraction
  a = a + b;
  b = a - b;
  a = a - b;
  printf("After swapping: a = %d, b = %d\n", a, b);
  return 0;
}
```

2. **Check if a Number is Positive, Negative, or Zero Using a Ternary Operator.**

```
Ans : #include <stdio.h>

int main() {

  int num;

  printf("Enter an integer: ");  //input from user

  scanf("%d", &num);

  (num > 0) ? printf("The number is positive.\n") :// // Checking the sign using nested ternary
operators

  (num < 0) ? printf("The number is negative.\n") :

        printf("The number is zero.\n");

  return 0;

}
```

3. **Write a program that checks whether a given number is even or odd using a ternary operator.**

Ans :
```
#include <stdio.h>

int main() {

   int num;

   // Input a number

   printf("Enter an integer: ");

   scanf("%d", &num);

   (num % 2 == 0) ? printf("The number is even.\n") : printf("The number is odd.\n");//checking
//even or odd by using ternary operator

   return 0;

}
```