Programming for Problem Solving
MST-1 [Solution]
ESE-104 [28th September, 2023]

1 Differentiate between semantic and logical errors with the help of example(s).

Semantic	**Logical**
1. Define: Semantic errors are the errors that occurred when the statements are not understandable by the compiler, errors in the meaning.	**Define:** The logical error is an error that leads to an undesired output. These errors produce the incorrect output, but they are error-free, known as logical errors.
2. Common mistakes are: 1. Type Mismatch 2. Uninitialized Variables. Example: 1. int num = "hello"; 2. int i; i=i+2;	**Common logical mistakes** 1. Incorrect Loop Conditions 2. Missing or Incorrect Increment/Decrement 3. Incorrect Order of Operations

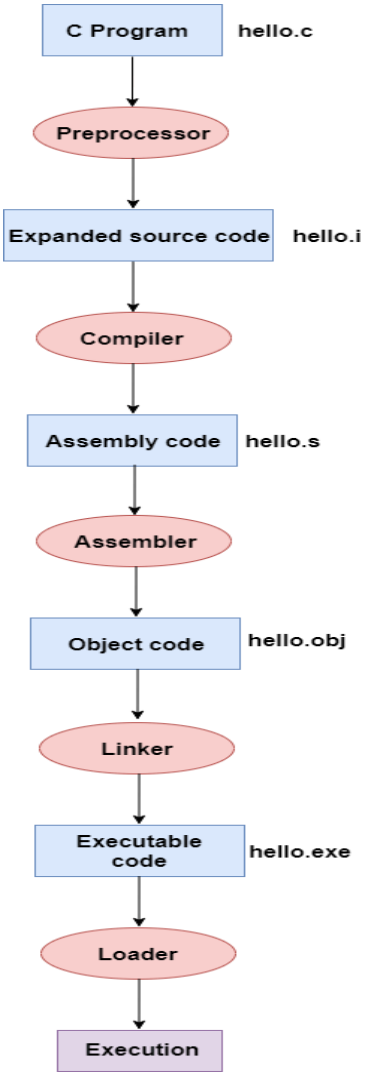
3. Code:
#include<stdio.h>
int main()
{
int a,b,c;
a=2;
b=3;
c=1;
a+b = c; // semantic error
return 0;
}
	Code:
	#include<stdio.h>
	int main()
	{
	for(int i=1; i<=10; i++); // logical error of putting semicolon after for loop
	{
	printf("%d", i);
	}
	return 0;
	}

2 What will be the output for the following code snippet:

```
int main ()
{
int i=3; j=3;
for(i=3; i<=5; i++)
  {
  for(j=3; j<=5; j++)
    {
    if(i = =4  && j = =4)
      {
      printf("I am applying continue statement");
       continue;
      }
      printf(" %d", i);
      printf(" %d", j);
        if(++i==6  || --j==7)

          {
              break;
          }
        }
```

	}
	printf(" %d",i);
	printf(" %d",j);
	return 0;
	}
	
	Output is: 3 3 4 3 5 3 7 3

3	Briefly illustrate the process of compilation and linking in C language with the help of diagram The following steps are taken to execute a program:  o Firstly, the input file, i.e., **hello.c,** is passed to the preprocessor, and the preprocessor converts the source code into expanded source code. The extension of the expanded source code would be **hello.i.** o The expanded source code is passed to the compiler, and the compiler converts this expanded source code into assembly code. The extension of the assembly code would be **hello.s.**

<table>
<tr><td></td><td>

This assembly code is then sent to the assembler, which converts the assembly code into object code.
After the creation of an object code, the linker creates the executable file. The loader will then load the executable file for the execution.

</td></tr>
<tr><td>4</td><td>Given a number N. Write a program to find sum of the digits in the number N. (For example N is 231, answer will be 6. Value of N should be read through the keyboard and case must be processed if N is positive integer only).

<pre>
#include <stdio.h>

int main() {
   int N, originalN, remainder, sum = 0;

   // Read a positive integer from the user
   printf("Enter a positive integer: ");
   scanf("%d", &N);

   // Check if N is a positive integer
   if (N <= 0) {
      printf("Please enter a positive integer.\n");
      return 1; // Exit the program with an error code
   }

   originalN = N; // Save the original value of N for later display

   // Calculate the sum of digits
   while (N > 0) {
      remainder = N % 10;
      sum += remainder;
      N /= 10;
   }

   // Display the result
   printf("The sum of digits in %d is: %d\n", originalN, sum);

   return 0;
}
</pre>
</td></tr>
<tr><td>5</td><td>Compare and contrast various loops.</td></tr>
</table>

For Loop	While Loop	Do-While
1. It is used when the number of iterations is known.	It is used when the number of iterations is not known.	A do-while loop is used where your loop should execute at least one time.
2. In case of no condition, the loop is repeated infinite times.	In case of no condition, an error will be shown.	If the condition is not put up in 'while' loop, it provides compilation error.
3.Initialization is not repeated.	Initialization is repeated if carried out during the stage of checking.	In do-while loop if initialization is done during condition checking, then initialization is done each time the loop iterate.
4.for(initialization; condition; iteration) { //body of 'for' loop }	while (condition) { //body of loop }	do { //body of the loop } while (testExpression);
5.Initialization can be in or out of the loop	Initialization is always out of the loop.	Initialization is always out of the loop.
6. //Print numbers from 1 to 5 #include<stdio.h> int main() { for(int i=1;i <= 5;i++) { printf("%d\n", i); } return 0; }	// Print numbers from 1 to 5 #include<stdio.h> int main() { int i = 1; while (i <= 5) { printf("%d\n", i); i++; } return 0; }	// Print numbers from 1 to 5 #include<stdio.h> int main() { int i = 1; do { printf("%d\n", i); i++; while (i <= 5) ; } return 0; }
7. **Real Life Example:** You were asked to walk for 20 minutes from the given time and you go on walking for the fixed time. There is a beginning and an end, all you need to do is continue your work till the destination is met.	**Real Life Example:** It is like filling a bucket with water using a mug. You never know how many mugs of water you use, but only be stopped when the bucket is full. Work until satisfying a given condition.	**Real Life Example:** Their are two examples 1. do { wash_hands; } while (hands_are_dirty); //continuously washing until completely washed. 2. do {

			eat( );
			//once you taste if good you continuously eat
			}
			while (eat_to_taste==good);

6 Design a menu driven code that does the following:

- If' l' is entered, user-defined function **'mul'** must be able to multiply two positive numbers u ng a user-defined function.
- If '2' is entered, user-defined function **'swap'** must be able to swap two numbers without using temporary variable.
- If *any* other integer is entered, code must be able to terminate with a suitable message.

(All types of inputs must be read through the keyboard with positive integers only.]

```
#include <stdio.h>

// Function prototypes
void multiply();
void swap();

int main() {
   char choice;

   do {
      // Display menu
      printf("Menu:\n");
      printf("1. Multiply two positive numbers\n");
      printf("2. Swap two numbers without a temporary variable\n");
      printf("3. Exit\n");
      printf("Enter your choice: ");
      scanf(" %c", &choice);  // Note the space before %c to consume the newline character

      // Perform actions based on user's choice
      switch (choice) {
         case '1':
            multiply();
            break;
         case '2':
            swap();
            break;
         case '3':
            printf("Exiting the program. Goodbye!\n");
            break;
         default:
            printf("Invalid choice. Please enter a valid option.\n");
      }
   }
while (choice != '3');
```

```
    return 0;
}

// Function to multiply two positive numbers
void multiply() {
    int num1, num2, result;

    // Input validation for positive integers
    do {
        printf("Enter the first positive number: ");
        scanf("%d", &num1);
    } while (num1 <= 0);

    do {
        printf("Enter the second positive number: ");
        scanf("%d", &num2);
    } while (num2 <= 0);

    // Perform multiplication
    result = num1 * num2;

    // Display the result
    printf("The product of %d and %d is: %d\n", num1, num2, result);
}

// Function to swap two numbers without using a temporary variable
void swap() {
    int num1, num2;

    // Input for two numbers
    printf("Enter the first number: ");
    scanf("%d", &num1);

    printf("Enter the second number: ");
    scanf("%d", &num2);

    // Swap without using a temporary variable
    num1 = num1 + num2;
    num2 = num1 - num2;
    num1 = num1 - num2;

    // Display the swapped values
    printf("After swapping: First number = %d, Second number = %d\n", num1, num2);
}
```