VIT®
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING
CAT 1, Feb 2024
B. Tech – Winter Semester 2023_2024

Course Code : BCSE102L **Duration** : 90 Minutes
Course Title : Structured and Object-Oriented Programming **Max. Marks:** 50
Slot : Common to C2 Slot

Answer Key

1.

a. C programming enables you to read or write data in a certain format. Explain any two formatted input and output statement with examples.

Formatted I/O functions are essential for handling user inputs and displaying outputs in a user-friendly way. They enable programmers to:

- Receive User Inputs: Formatted input functions help programs collect user input in a structured manner. They use format specifiers to interpret and extract specific data types from user input.
- Present Data to Users: Formatted output functions allow programs to present data to users in various formats. Format specifiers are used to control how data is displayed, making it more readable and visually appealing.
- Support Multiple Data Types: These I/O functions are versatile and support a wide range of data types, including integers, floating-point numbers, characters, and more. Each data type has corresponding format specifiers for precise formatting.
- Formatting Control: Programmers can use format specifiers to control the alignment, width, precision, and other formatting aspects of displayed data, ensuring it's presented as intended.

Some of the widely used formatted input and output statements are

1. printf()
2. scanf()
3. sprintf()
4. sscanf()

1. **printf():**

In C, printf() is a built-in function used to display values like numbers, characters, and strings on the console screen. It's pre-defined in the stdio.h header, allowing easy output formatting in C programs.

Syntax 1

printf(“Format Specifier”, var1, var2, …., varn);

Example

```
// C program to implement printf() function
#include <stdio.h>
 int main()
{
   int a;
   a = 20;
   printf("%d", a);
   return 0;
}
```

Output

20

2. **scanf():**

- **scanf():** In C, scanf() is a built-in function for reading user input from the keyboard. It can read values of various data types like integers, floats, characters, and strings. scanf() is a pre-defined function declared in the stdio.h header file. It uses the & (address-of operator) to store user input in the memory location of a variable.

Syntax

scanf(“Format Specifier”, &var1, &var2, …., &varn);

Example

```
// C program to implement  scanf() function
#include <stdio.h>
int main()
{
   int num1;
   printf("Enter a integer number: ")
   scanf("%d", &num1);
   printf("You have entered %d", num1);
   return 0;
}
```

Output

Enter a integer number: You have entered 0

Output

Enter a integer number: 56

You have entered 56

3. **sprintf():**

sprintf(): Short for “string print,” sprintf() is similar to printf() but it stores the formatted string into a character array instead of displaying it on the console screen.

Syntax

```
// C program to implement the sprintf() function

#include <stdio.h>

int main()

{

   char str[50];

   int a = 2, b = 8;

   sprintf(str, "%d and %d are even number",  a, b);


   printf("%s", str);

   return 0;

}
```

Output

2 and 8 are even number

4. **sscanf():**

- **sscanf():** Abbreviated for “string scanf,” sscanf() resembles scanf() but reads data from a string or character array rather than from the console screen.

Syntax

sscanf(array_name, “format specifier”, &variable_name);

Example

```
// C program to implement sscanf() function

#include <stdio.h>

int main()

{

   char str[50];

   int a = 2, b = 8, c, d;
```

```
  sprintf(str, "a = %d and b = %d",          a, b);
 sscanf(str, "a = %d and b = %d",
      &c, &d);
 printf("c = %d and d = %d", c, d);
 return 0;
}
```

Output

c = 2 and d = 8

b. Explain the methods with example for Interrupting Loop Flow in the context of C programming

Interruptions in the loop flow in C can be carried out using Break and continue

1. break Statement:
    - The break statement is used to terminate the execution of a loop prematurely.
    - When encountered within a loop (such as for, while, or do-while), it immediately exits the loop, regardless of the loop condition.
    - Common use cases:
        - Breaking out of an infinite loop when a specific condition is met.
        - Terminating a loop early based on some criteria.
    - Example:

```
#include<stdio.h>
Void main()
{
for (int i = 1; i <= 10; i++) {
   if (i == 5) {
      break; // Exit the loop when i reaches 5
   }
   printf("%d ", i);
}
// Output: 1 2 3 4
```

2. continue Statement:
    - The continue statement is used to skip the current iteration of a loop and move to the next iteration.
    - When encountered, it jumps directly to the loop's control expression (e.g., the increment/decrement part of a for loop).
    - Common use cases:
        - Skipping specific iterations based on certain conditions.
        - Continuing with the next iteration even if some intermediate steps are unnecessary.

    Example:

```
#include<stdio.h>

Void main()

{
```

```
for (int i = 1; i <= 10; i++) {
   if (i % 2 == 0) {
      continue; // Skip even numbers
   }
   printf("%d ", i);
}
}
// Output: 1 3 5 7 9
```

2. Explain if, if-else, nested if-else and cascaded if-else with examples and syntax.

The conditional constructs allow us to control the flow of our program based on certain conditions. Here are the different types of conditional statements, along with examples and syntax:

if Statement:

The if statement is the most straightforward decision-making construct. It allows us to execute a block of code if a specified condition is true.

Syntax:

```
if (condition) {
   // Statements to execute if the condition is true
}
```

Example:

```
#include <stdio.h>
int main() {
   int i = 10;
   if (i > 15) {
      printf("10 is greater than 15");
   }
   printf("I am Not in if");
   return 0;
}
```

Output:

I am Not in if

In this example, since the condition (i > 15) is false, the block inside the if statement is not executed.

if-else Statement:

The if-else statement allows us to execute different blocks of code based on whether a condition is true or false.

Syntax:

```
if (condition) {
   // Executes this block if the condition is true
} else {
   // Executes this block if the condition is false
}
```

Example:

```
#include <stdio.h>
int main() {
   int age = 20;
   if (age >= 18) {
      printf("You are eligible to vote!");
   } else {
      printf("You are not eligible to vote.");
   }
   return 0;
}
```

Output:

You are eligible to vote!

Nested if-else Statement:

We can nest if-else statements inside each other. This allows us to handle multiple conditions.

Example:

```
#include <stdio.h>
int main() {
   int score = 75;
   if (score >= 90) {
      printf("Excellent!");
   } else if (score >= 80) {
      printf("Good!");
   } else if (score >= 70) {
      printf("Average.");
   } else {
      printf("Needs improvement.");
```

```
    }
    return 0;
}
```

Output:

Average.

Cascaded if-else-if Ladder:

The cascaded if-else-if ladder is used when we need to decide among multiple options. Conditions are evaluated sequentially, and the first true condition's associated block is executed.

Example:

```
#include <stdio.h>
int main() {
    int day = 3;
    if (day == 1) {
        printf("Monday");
    } else if (day == 2) {
        printf("Tuesday");
    } else if (day == 3) {
        printf("Wednesday");
    } else {
        printf("Invalid day");
    }
    return 0;
}
```

Output:

Wednesday

3.

a. Write a c function str_concat() to Concatenate Two Strings Without Using strcat() function available in string.h header

```
#include <stdio.h>
void str_concat(char dest[], const char src[]) {
    int i, j;
    // Find the end of the destination string
```

```
    for (i = 0; dest[i] != '\0'; i++);
    // Concatenate the source string to the destination
    for (j = 0; src[j] != '\0'; j++, i++) {
        dest[i] = src[j];
    }
    // Add the null terminator
    dest[i] = '\0';
}
int main() {
    char str1[100], str2[100];
    printf("Enter the first string: ");
    gets(str1);

    printf("Enter the second string: ");
    gets(str2);
    str_concat(str1, str2);
    printf("Concatenated string: %s\n", str1);
    return 0;
}
// Output
Enter the first string: hello
Enter the second string: world
Concatenated string: helloworld
```

c. Write a program in C to find two elements whose sum is closest to zero. The given array is : 38 44 63 -51 -35 19 84 -69 4 -46

```
The Pair of elements whose sum is minimum are: [44, -46]
#include <stdio.h>
#include <stdlib.h>

void minAbsSumPair(int arr[], int arr_size) {
    int min_l = 0, min_r = 1;
    int min_sum = arr[0] + arr[1];
```

```
    for (int l = 0; l < arr_size - 1; l++) {
        for (int r = l + 1; r < arr_size; r++) {
            int sum = arr[l] + arr[r];
            if (abs(sum) < abs(min_sum)) {
                min_sum = sum;
                min_l = l;
                min_r = r;
            }
        }
    }

    printf("The two elements whose sum is minimum are %d and %d\n", arr[min_l], arr[min_r]);
}

int main() {
    int arr[] = {38, 44, 63, -51, -35, 19, 84, -69, 4, -46};
    int arr_size = sizeof(arr) / sizeof(arr[0]);

    printf("The given array is: ");
    for (int i = 0; i < arr_size; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");

    printf("The Pair of elements whose sum is minimum are: ");
    minAbsSumPair(arr, arr_size);

    return 0;
}
```

Output:

The Pair of elements whose sum is minimum are: 44 and -46

4. a. Write a C function that models the following mathematical function. For example, f(-3.2) returns -3.2 + 2 = -1.2.

Assume that x is a float type variable.

$$f(x) = \begin{cases} x + 2, & x < -1 \\ x2-, & -1 \le x \le 1 \\ -x + 2, & x > 1 \end{cases}$$

```
#include <stdio.h>

#include <stdlib.h>

float f(float x)

{

 if (x < -1)

 return x + 2;

 else if ((x >= -1) && (x <= 1))

 return x * x;

 else if (x > 1)

 return -x + 2;

}

int main()

{   float num, res;

   scanf("%f",&num);

   res=f(num);

   printf("%f",res);

   return 0;

}
```

b. Write a Program to Generate the Fibonacci Series 0, 1, 1, 2, 3, 5, 8……using recursion

```
#include <stdio.h>

// Function to calculate the nth Fibonacci number
int fibonacci(int num) {
   // Base condition
   if (num == 0 || num == 1) {
      return num;
   } else {
      // Recursive call
      return fibonacci(num - 1) + fibonacci(num - 2);
```

```
    }
}

int main() {
    int terms;

    printf("Enter the number of terms: ");
    scanf("%d", &terms);

    if (terms < 1) {
        printf("Invalid number of terms\n");
    } else {
        printf("Fibonacci series up to %d terms: ", terms);
        for (int n = 0; n < terms; n++) {
            printf("%d ", fibonacci(n));
        }
        printf("\n");
    }

    return 0;
}
```

5

```
#include <stdio.h>
#include <stdlib.h>

void removeDuplicateBookAccessNumber(int* bookAccessNumbers, int* n) {
    int i, j, k;
    int size = *n;

    // Find duplicate values in the array
    for (i = 0; i < size; i++) {
        for (j = i + 1; j < size; j++) {
            // If any duplicate is found
            if (bookAccessNumbers[i] == bookAccessNumbers[j]) {
                // Delete the current duplicate element
                for (k = j; k < size - 1; k++) {
                    bookAccessNumbers[k] = bookAccessNumbers[k + 1];
                }
```

```
            size--;
            j--;
          }
        }
    }

    *n = size;
}

int main(void) {
    int bookAccessNumbers[] = {1111, 5656, 1111, 8956, 2222, 6546, 2222, 56895, 2222, 50565};
    int size = 10, i;

    printf("Book access numbers array before removing duplicates:\n");
    for (i = 0; i < size; i++) {
        printf("%d, ", bookAccessNumbers[i]);
    }

    removeDuplicateBookAccessNumber(bookAccessNumbers, &size);

    printf("\n\nBook access numbers array after removing duplicates:\n");
    for (i = 0; i < size; i++) {
        printf("%d, ", bookAccessNumbers[i]);
    }

    getchar(); // Wait for user input
    getchar(); // Wait for another input (optional)
    return 0;
}
```

b. Elaborate in the context of c programming to Return multiple value from function – using pointers with examples

In a function though call by reference method can be used to pass any number of parameters as reference . Using call by reference, we can make function return more than one values at a time

```
#include<stdio.h>

void circleparams(int, float*, float*);

void main()

{

 int radius;

 float area, perimeter;

 printf("\nEnter radius of a circle: ");

 scanf("%d", &radius);

 circleparams(radius, &area, &perimeter);

 printf("\nCircle Area: %f", area);

 printf("\nCircle Perimeter: %f", perimeter);

}


void circleparams(int cradius, float *carea, float *cperimeter)

{

 *carea = 3.14 * cradius * cradius;

 *cperimeter = 2 * 3.14 * cradius;

}
```