GUJARAT TECHNOLOGICAL UNIVERSITY

BE- SEMESTER–1 PAPER SOLUTION – WINTER 2024

Subject Name & Code:
Programming for Problem Solving- BE01000121

Marks

Q.1 **(a)** Write an algorithm to check whether the entered number is Even or Odd. **03**

Answer:

Algorithm to Check Whether a Number is Even or Odd
Step 1: Start
Step 2: Input the number
Step 3: If (number % 2 == 0) → Print "Even"
Step 4: Else → Print "Odd"
Step 5: Stop

(b) Describe various symbols used for preparing flow chart. **04**

Answer:

Symbol	Name	Description
Oval	Terminal	Start or End of the program
Parallelogram	Input/Output	Represents input or output operation
Rectangle	Process	Shows a process or operation
Diamond	Decision	Used for decision-making (Yes/No, True/False)
Arrow	Flow Line	Shows direction of flow

(c) Explain different type of operators used in c language with their precedence and associativity. **07**

Answer:

Operators in C Language with Precedence and Associativity
Operators are symbols that perform operations on variables and values. They are categorized as:
Types of Operators:

1. **Arithmetic Operators:** +, -, *, /, %
2. **Relational Operators:** ==, !=, >, <, >=, <=
3. **Logical Operators:** &&, ||, !
4. **Assignment Operators:** =, +=, -=, *=, /=, %=
5. **Increment/Decrement:** ++, --

6. **Conditional Operator:** ? :
7. **Bitwise Operators:** &, |, ^, ~, <<, >>
8. **Special Operators:** sizeof, &, * (pointer)

Operator Precedence and Associativity Table:

Precedence Level	Operators	Associativity
Highest	++, --, (), [], .	Left to Right
	+, - (Unary), !, ~	Right to Left
	*, /, %	Left to Right
	+, -	Left to Right
	<, <=, >, >=	Left to Right
	==, !=	Left to Right
	&&, `	
	?:	Right to Left
	=, +=, -= etc.	Right to Left
Lowest	,	Left to Right

Q.2 **(a)** Differentiate: Exit controlled Loop and Entry controlled Loop **03**

Answer:

Entry Controlled Loop	Exit Controlled Loop
Condition is **checked before** loop executes.	Condition is **checked after** loop executes.
It may execute **zero or more times**.	Executes **at least once** regardless of condition.
Example: `for`, `while` loop.	Example: `do...while` loop.

(b) Explain: **04**
a) break statement
b) continue statement

Answer:

Explain:
a) break statement:
- Used to **exit** from loops or switch cases immediately.
- Control comes out of the nearest enclosing loop or block.

Example:

```c
for (int i = 0; i < 5; i++) {
    if (i == 3)
        break;
    printf("%d ", i);
}
// Output: 0 1 2
```

b) continue statement:

- Skips the **rest of the loop body** and proceeds with the **next iteration**.
- Used to skip certain values in loops.

Example:

```c
for (int i = 0; i < 5; i++) {
    if (i == 2)
        continue;
    printf("%d ", i);
}
// Output: 0 1 3 4
```

(c) Write a C program to find factorial of a number. **07**

Answer:

Explanation:

Factorial of n is defined as:

$$n! = n \times (n-1) \times (n-2) \times \cdots \times 1$$

Code:

```c
#include <stdio.h>

int main() {
    int num, i;
    unsigned long long fact = 1;

    printf("Enter a positive integer: ");
    scanf("%d", &num);

    if (num < 0)
        printf("Factorial is not defined for negative numbers.");
    else {
        for (i = 1; i <= num; i++) {
            fact *= i;
        }
        printf("Factorial of %d = %llu", num, fact);
    }
    return 0;
}
```

OR

(c) Construct a C program to swap the values of two integers with use of only two variables. **07**

Answer:

Swap Two Integers Using Only Two Variables
Concept:
Use arithmetic operations without needing a third temporary variable.
Code:

```c
#include <stdio.h>

int main() {
    int a, b;

    printf("Enter two integers: ");
    scanf("%d %d", &a, &b);
```

```c
    // Swapping using arithmetic
    a = a + b;
    b = a - b;
    a = a - b;

    printf("After swapping: a = %d, b = %d", a, b);
    return 0;
}
```

Example Input/Output:
Input: a = 4, b = 5
Output: a = 5, b = 4

Q.3 **(a)** Demonstrate the use of conditional operator statement with an example. **03**

Answer:

```c
(condition) ? expression_if_true : expression_if_false;
```

```c
#include <stdio.h>

int main() {
    int num = 10;
    char *result;

    result = (num % 2 == 0) ? "Even" : "Odd";
    printf("The number is %s", result);

    return 0;
}
```

Output:

```
The number is Even
```

(b) Explain the structure of switch case statement. **04**

Answer:

Structure of switch Case Statement
Purpose:
Used for multi-way decision-making based on a variable's value.

```
switch (expression) {
    case constant1:
        // statements
        break;
    case constant2:
        // statements
        break;
    ...
    default:
        // default statements
}
```

Example:

```
int day = 2;
switch (day) {
    case 1: printf("Monday"); break;
    case 2: printf("Tuesday"); break;
    default: printf("Other Day");
```

Output:

```
Tuesday
```

(c) Write a program to print the triangle shown below. **07**

1
1 0
1 0 1
1 0 1 0
1 0 1 0 1

Answer:

Code:

```
#include <stdio.h>

int main() {
    int i, j;

    for (i = 1; i <= 5; i++) {
        for (j = 1; j <= i; j++) {
            if (j % 2 == 1)
                printf("1 ");
            else
                printf("0 ");
        }
        printf("\n");
    }

    return 0;
}
```

OR

Q.3 **(a)** Write a program to check whether entered character is vowel or not? **03**

Answer:
Code:

```c
#include <stdio.h>

int main() {
    char ch;

    printf("Enter a character: ");
    scanf("%c", &ch);

    if (ch == 'a' || ch == 'e' || ch == 'i' ||
        ch == 'o' || ch == 'u' ||
        ch == 'A' || ch == 'E' || ch == 'I' ||
        ch == 'O' || ch == 'U')
        printf("%c is a vowel.", ch);
    else
        printf("%c is not a vowel.", ch);

    return 0;
}
```

(b) Explain getch(), getchar(), gets(), puts() . **04**

Answer:

Explain: getch(), getchar(), gets(), puts()

Function	Description
`getch ()`	Reads a character from keyboard, **does not echo** to screen.
`get char()`	Reads a single character from input, **echoes** to screen.
`gets ()`	Reads a **string** from user until a newline (`\n`) is found.
`puts ()`	Displays a string to screen with **automatic newline**.

(c) Write a program to add all the elements of an array. **07**

Answer:
Code:

```c
#include <stdio.h>

int main() {
    int arr[100], n, sum = 0;

    printf("Enter number of elements: ");
    scanf("%d", &n);

    printf("Enter %d elements:\n", n);
    for (int i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
        sum += arr[i];
    }

    printf("Sum of all elements = %d", sum);
    return 0;
}
```

Q.4 **(a)** What is structure? Explain with example how to declare a structure and how to initialize it. **03**

Answer:

A **structure** is a user-defined data type in C that allows grouping of variables of different types under one name.

Declaration Syntax:

```c
struct Student {
    int roll;
    char name[30];
    float marks;
};
```

Initialization Example:

```c
struct Student s1 = {101, "Amit", 87.5};
```

You can also assign values after declaration:

```c
strcpy(s1.name, "Amit");
s1.roll = 101;
s1.marks = 87.5;
```

(b) Explain following string manipulation function. **04**
strcat( ), strcpy( ) ,strcmp( ) and strlen( )

Answer:

1. `strcat(s1, s2)` – Appends string `s2` to the end of string `s1`.

```c
char s1[20] = "Hello ";
char s2[10] = "World";
strcat(s1, s2);  // s1 = "Hello World"
```

2. `strcpy(dest, src)` – Copies string `src` into `dest`.

```c
char dest[20];
strcpy(dest, "GTU");  // dest = "GTU"
```

3. `strcmp(s1, s2)` – Compares two strings lexicographically.
Returns 0 if equal, <0 if s1<s2, >0 if s1>s2.

```c
strcmp("abc", "abc"); // returns 0
```

4. `strlen(s)` – Returns the length of the string `s`.

```c
strlen("Hello") → 5
```

(c) Define a Structures which contains details of a Cricketer: **07**
- Name of Player:
- Team name:
- Total run scored:
- Batting average:

Answer:
Structure Definition and Program:

```c
#include <stdio.h>
#include <string.h>

struct Cricketer {
    char name[30];
    char team[20];
    int totalRuns;
    float battingAvg;
};

int main() {
    struct Cricketer player;
```

```c
int main() {
    struct Cricketer player;

    // Input
    printf("Enter Player Name: ");
    gets(player.name);
    printf("Enter Team Name: ");
    gets(player.team);
    printf("Enter Total Runs: ");
    scanf("%d", &player.totalRuns);
    printf("Enter Batting Average: ");
    scanf("%f", &player.battingAvg);
```

```c
    // Output
    printf("\nPlayer Details:\n");
    printf("Name: %s\n", player.name);
    printf("Team: %s\n", player.team);
    printf("Total Runs: %d\n", player.totalRuns);
    printf("Batting Average: %.2f\n", player.battingAvg);

    return 0;
}
```

OR

Q.4 **(a)** Debug the following codes and show the output for this: **03**

```
For ( i=0 ; i<=50 ; i++ );
{
printf( "\n %d" , i )
}
```

*consider header files and declaration of variable is done.

Answer:

Debug and Output the Code

```c
For ( i = 0 ; i <= 50 ; i++ );  // ← Semicolon here is wrong
{
   printf("\n %d" , i );
}
```

Corrected Code:

```c
for (i = 0; i <= 50; i++) {
    printf("\n %d", i);
}
```

Error Explanation:

- Semicolon ; after for ends the loop prematurely.
- {} block runs **only once**, and i is out of scope if not declared globally.

(b) Summarize the methods for initialization of One-Dimensional array. **04**

Answer:

1. **At Declaration:**

```c
int arr[5] = {10, 20, 30, 40, 50};
```

2. **Partial Initialization:**

```c
int arr[5] = {10, 20}; // Remaining elements become 0
```

3. **Runtime Input via Loop:**

```c
int arr[5], i;
for (i = 0; i < 5; i++)
    scanf("%d", &arr[i]);
```

4. **Without Size (auto):**

```c
int arr[] = {5, 10, 15}; // size becomes 3
```

(c) Construct a C program to add 3X3 matrix. **07**

Answer:

C Program to Add Two 3×3 Matrices

```c
#include <stdio.h>

int main() {
    int a[3][3], b[3][3], sum[3][3];
    int i, j;

    printf("Enter elements of Matrix A:\n");
    for (i = 0; i < 3; i++)
        for (j = 0; j < 3; j++)
            scanf("%d", &a[i][j]);

    printf("Enter elements of Matrix B:\n");
    for (i = 0; i < 3; i++)
        for (j = 0; j < 3; j++)
            scanf("%d", &b[i][j]);
```

```c
// Matrix addition
for (i = 0; i < 3; i++)
    for (j = 0; j < 3; j++)
        sum[i][j] = a[i][j] + b[i][j];
```

```c
    // Display result
    printf("Sum of Matrices:\n");
    for (i = 0; i < 3; i++) {
        for (j = 0; j < 3; j++)
            printf("%d ", sum[i][j]);
        printf("\n");
    }

    return 0;
}
```

Q.5 **(a)** Differentiate between structure and union. **03**

Answer:

Feature	Structure	Union
Memory Allocation	Each member has **separate memory**	All members **share the same memory**
Access	All members can be accessed simultaneously	Only **one member at a time**
Size	Sum of sizes of all members	Size of **largest member only**
Use Case	When storing multiple **independent** values	When storing **one of many** values

(b) Explain fopen() and its mode with example. **04**

Answer:

Explain `fopen()` and Its Modes with Example

`fopen()` is used to open a file in C for reading or writing.

Syntax:

```c
FILE *fp;
fp = fopen("filename.txt", "mode");
```

Mode	Description
`"r"`	Read mode (file must exist)
`"w"`	Write mode (creates new or overwrites file)
`"a"`	Append mode (adds to end if file exists)
`"r+"`	Read + write (file must exist)
`"w+"`	Read + write, overwrites or creates new

Example:

```c
FILE *fp;
fp = fopen("data.txt", "w");
fprintf(fp, "Hello, GTU!");
fclose(fp);
```

(c) Briefly explain any two file handling functions with an example. **07**

Answer:

Two File Handling Functions with Example
1. fopen() – Opens a file.
2. fclose() – Closes a file.
3. fprintf() – Writes formatted output to a file.
4. fscanf() – Reads formatted input from a file.

```c
#include <stdio.h>

int main() {
    FILE *fp;

    // Writing to file
    fp = fopen("test.txt", "w");
    if (fp == NULL) {
        printf("File not created.");
        return 1;
    }
    fprintf(fp, "GTU File Handling Example\n");
    fclose(fp);
```

```c
    // Reading from file
    char str[100];
    fp = fopen("test.txt", "r");
    fgets(str, 100, fp);
    printf("File Content: %s", str);
    fclose(fp);

    return 0;
}
```

OR

Q.5 **(a)** Write True / False against following statements: **03**

1) Pointer is a variable which holds address of location where value of other variable is stored.
2) Pointer holds the address in form of float or integer.
3) The address of a variable can be stored in two or more pointers.

Answer:

True / False Statements

1. **Pointer is a variable which holds address of location where value of other variable is stored.**
   ✅ **True**
2. **Pointer holds the address in form of float or integer.**
   ❌ **False** – Pointers hold addresses (not values), but their **type** (e.g., int*, char*) defines the data type they point to.
3. **The address of a variable can be stored in two or more pointers.**
   ✅ **True**

(b) Write a C program to copy content of one file to other with the help of file handling functions. **04**

Answer:

Program to Copy Content from One File to Another

```c
#include <stdio.h>

int main() {
    FILE *f1, *f2;
    char ch;

    f1 = fopen("source.txt", "r");
    f2 = fopen("destination.txt", "w");

    if (f1 == NULL || f2 == NULL) {
        printf("Error opening file.");
        return 1;
    }
while ((ch = fgetc(f1)) != EOF)
    fputc(ch, f2);
```

```
        printf("File copied successfully.");
        fclose(f1);
        fclose(f2);

        return 0;
    }
```

(c) Develope a program in C to check the entered number is prime or not by creating a user-defined function named check_prime(). **07**

Answer:

C Program to Check if a Number is Prime using check_prime() Function

```
#include <stdio.h>

// User-defined function
int check_prime(int num) {
    if (num <= 1) return 0;
    for (int i = 2; i <= num / 2; i++) {
        if (num % i == 0)
            return 0;  // Not prime
    }
    return 1;  // Prime
}
```

```
int main() {
    int number;
    printf("Enter a number: ");
    scanf("%d", &number);

    if (check_prime(number))
        printf("%d is a prime number.", number);
    else
        printf("%d is not a prime number.", number);

    return 0;
}
```

****