SGK GOVERNMENT DEGREE COLLEGE

VINUKONDA, PALNADU DISTRICT

Report on Remedial Coaching

for the Course Programming in C

Academic Year 2021-22

CONDUCTED BY

DEPARTMENT OF COMPUTER SCIENCE

The remedial coaching program for "Programming in C" was designed to cater to the specific needs of slow learners who scored less than 15 internal marks. Out of the 38 students enrolled, 4 were identified as slow learners. After the coaching, 2 out of the 4 slow learners showed improvement, achieving a total of more than 40 marks.

The following is the Classification of students into Slow Learners, Medium Learners and Advanced Learners.

S.No	REGD.NO	NAME OF THE STUDENT	Internal Marks	EXTERNAL MARKS	Total	Learner Classification
1	Y213099048	BAILADUGU TARAKA ARJUN	19	31	50	Medium Learner
2	Y213099049	BANKA GURAVAMMA	23	40	63	Advanced Learner
3	Y213099050	CHANGALA SAI VENKATA GANESH	16	30	46	Medium Learner
4	Y213099051	DASARI RATNA KUMARI	19	8	27	Medium Learner
5	Y213099052	GOPU VENKATESH	23	33	56	Advanced Learner
6	Y213099053	JEEDIMALLA BHANU PRASAD	22	49	71	Advanced Learner
7	Y213099054	KAKANI BRAHMA NAIDU	10	6	16	Slow Learner
8	Y213099055	KANCHARLA GAYATHRI	23	35	58	Advanced Learner
9	Y213099056	KISTAM CHANDRA SEKHAR	13	0	13	Slow Learner
10	Y213099057	KOTA VENKATA LAKSHMI PADMAVATHI	23	11	34	Advanced Learner
11	Y213099058	MALAPATI KRUPARAJU	20	47	67	Advanced Learner
12	Y213099059	MANNEPALLI VENKATESH	19	36	55	Medium Learner
13	Y213099060	MUTUKURI KIRAN KUMAR	15	45	60	Medium Learner
14	Y213099061	NAMBURI PRADEEP	10	A	A	Slow Learner
15	Y213099062	ORSU GOPI	16	6	22	Medium Learner
16	Y213099063	PADARA VENKAT	19	4	23	Medium Learner
17	Y213099064	PALADUGU MAHESWARI	22	49	71	Advanced Learner
18	Y213099065	PALLEPOGU RAKESH	16	10	26	Medium Learner
19	Y213099066	PASUMARTHI RAHIMUNNISA	23	41	64	Advanced Learner
20	Y213099067	PEDDEETI DURGA PRASAD	19	2	21	Medium Learner
21	Y213099068	SETTI NAGALAKSHMAIAH	17	5	22	Medium Learner
22	Y213099069	SHAIK RAAFIYA	23	35	58	Advanced Learner
23	Y213099070	SHAIK ZAKIRA ROSHAN	24	43	67	Advanced Learner
24	Y213099071	SHIAK IMRAN BASHA	22	39	61	Advanced Learner
25	GG	VELPULA AKSHAYA BABU	19	4	23	Medium Learner
26	Y213099073	YADLAPALLI VASU	17	54	71	Medium Learner
27	Y213099074	YESUPOGU MOUNIKA	10	11	21	Slow Learner
28	Y213099075	AMRUTHAPUDI BHASKAR RAO	23	50	73	Advanced Learner
29	Y213099076	BANDARU VEERANJANEYULU	23	20	43	Advanced Learner
30	Y213099077	BHAVANASI BHARATHI	16	43	59	Medium Learner

31	Y213099078	IRIGI DEVAKUMARI	23	48	71	Advanced Learner
32	Y213099079	KAMBHAMPATI VEERA BRAHMA CHARI	16	32	48	Medium Learner
33	Y213099080	KOTWAL VAHEEDAREHAMAN	18	30	48	Medium Learner
34	Y213099081	PILLIKUDUPULA SRINIVASA GOPI	10	30	40	Slow Learner
35	Y213099082	SHAIK JANBEE	19	40	59	Medium Learner
36	Y213099083	SOMAVARAPU YESHAYA	24	62	86	Advanced Learner
37	Y213099084	SRIKAKULAM NAGENDRACHARI	21	64	85	Advanced Learner
38	Y213099085	VALLEM VENKATESWARLU	10	30	40	Slow Learner

List of topics taught during remedial coaching:

S.No	Name of the Topic	Brief Synopsis of the Topic
1	Introduction to C	C is a widely-used, versatile programming language renowned for its efficiency and direct memory access. It was developed by Dennis Ritchie at Bell Labs in the early 1970s. C serves as a foundational language for many others, making it a crucial starting point for programmers. It provides low-level memory manipulation and access, making it powerful yet challenging, especially for beginners.
2	Variables and Data Types	Variables are storage locations for holding data during program execution. In C, understanding various data types (integers, floating-point numbers, characters, etc.) and their characteristics is essential. Variables and data types are fundamental concepts forming the basis of any C program.

3	Control Flow and Loops	Control flow structures enable the program to make decisions and execute specific code blocks accordingly. Loops, including for, while, and do-while, facilitate repetitive execution, allowing efficient program flow and reducing redundancy in code.
4	Functions	Functions are blocks of organized and reusable code aimed at performing specific tasks. They enhance code modularity, readability, and reusability, crucial for managing complex programs.
5	Arrays	Arrays in C are collections of elements, all of the same data type, accessed using an index. They provide an efficient way to store and manipulate multiple values under a single variable name.
6	Pointers	Pointers are variables that store memory addresses. Understanding pointers is critical for dynamic memory allocation, efficient array manipulation, and interfacing with hardware.
7	Structures and Unions	Structures allow combining different data types into a single entity, while unions enable storing different data types in the same memory location. Both are essential for organizing and handling complex data.

8	File Input/Output	File I/O functions in C enable reading from and writing to files, a fundamental aspect for data persistence and management.
9	Dynamic Memory Allocation/Deallocation	Dynamic memory allocation allows a program to request memory during runtime. It's a crucial concept for managing memory effectively, especially when the amount of memory required is unknown or varying. Deallocation is equally important to release memory and prevent memory leaks.
10	Preprocessor Directives and Macros	Preprocessor directives provide instructions to the compiler before actual compilation. Macros are a way to define constants or short functions, enhancing code maintainability and readability.

Beneficiary Status of Slow Learners:

S.No	**REGD. NO**	**NAME OF THE STUDENT**	**Internal Marks**	**EXTERNAL MARKS**	**Total**	**Learner Classification**	**Outcome of Remedial Coacing**
1	Y213099054	KAKANI BRAHMA NAIDU	10	6	16	Slow Learner	Not Benefitted
2	Y213099074	YESUPOGU MOUNIKA	10	11	21	Slow Learner	Not Benefitted
3	Y213099081	PILLIKUDUPULA SRINIVASA GOPI	10	30	40	Slow Learner	Benefitted
4	Y213099085	VALLEM VENKATESWARLU	10	30	40	Slow Learner	Benefitted

Overview of Topics planned for Remedial Coaching

1. Introduction to Programming in C

Description: C programming is the bedrock of modern computing and serves as a foundational language for both system-level and application-level programming. Originating at Bell Laboratories in the early 1970s by Dennis Ritchie, C has stood the test of time and is widely utilized due to its versatility and efficiency.

In C, a program comprises functions, which manipulate data and perform specific tasks. Functions are integral to C programming, enabling modularity, code reusability, and a structured approach to problem-solving. These functions can return values and can be invoked within the program, enhancing the program's clarity and efficiency.

Example:

```
#include <stdio.h>


int main() {

    printf("Hello, World!");

    return 0;

}
```

In this example, we use the **printf** function to display "Hello, World!" on the standard output. The **main** function, as the entry point of a C program, orchestrates the program's execution.

2. Variables and Data Types

Description: Variables in C are fundamental units that store data temporarily during program execution. Every variable has a specific data type, such as int (integer), char (character), float (floating-point), and double (double-precision floating-point), which dictates the kind of value it can hold.

Understanding variables and data types is essential for effective memory usage and ensuring program correctness. Proper usage ensures that the right amount of memory is allocated for different types of data, preventing issues like overflow or data loss.

Example:

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

```
int main() {
    int age = 30;
    char gender = 'M';
    float height = 5.9;

    printf("Age: %d\n", age);
    printf("Gender: %c\n", gender);
    printf("Height: %.2f\n", height);

    return 0;
```

}In this example, we declare variables of different data types (int, char, float) and display their values using the **printf** function.

3. Control Structures: Loops and Conditionals

Description: Control structures in C govern the flow of execution within a program. They include conditionals (if-else) and loops (for, while, do-while). Conditionals allow branching based on certain conditions, while loops enable repetitive execution of a block of code.

Understanding control structures is vital for creating efficient and adaptable programs capable of handling various scenarios and processing data accordingly.

Example:

```
#include <stdio.h>

int main() {
    int i;
    for(i = 1; i <= 10; i++) {
        printf("%d ", i);
    }
    if(i == 11) {
        printf("\nLoop executed successfully.\n");
    } else {
        printf("\nLoop failed.\n");
    }

    return 0;
}
```

In this example, we use a **for** loop to print numbers from 1 to 10 and then use an **if-else** statement to check if the loop executed successfully.

4. Functions and Modularity

Description: Functions in C are blocks of code that can be reused and called within a program. They enhance modularity, making it easier to manage and comprehend the codebase. Functions accept inputs (parameters), perform specific tasks, and may return a value.

Understanding functions and their usage is fundamental for writing efficient, maintainable, and modular code. Functions allow for code reuse and aid in organizing the program's logic into manageable units.

Example:

```
#include <stdio.h>

int add(int a, int b) {
    return a + b;
}

int main() {
    int num1 = 10, num2 = 20;
    int sum = add(num1, num2);
    printf("Sum: %d", sum);

    return 0;
}
```

In this example, we define a function **add** that calculates the sum of two integers. The **main** function calls this function to obtain the sum of **num1** and **num2**.

5. Arrays and Strings

Description: Arrays in C allow for the storage of multiple values of the same data type under a single name. Arrays play a crucial role in handling collections of data efficiently. Strings, essentially, are arrays of characters.

Understanding arrays and strings is essential for effective data handling and manipulation in C programs. Arrays provide a structured way to store and access elements of the same type, aiding in managing and organizing data.

Example:

```
#include <stdio.h>

int main() {
    int numbers[5] = {10, 20, 30, 40, 50};
    char greeting[6] = "Hello";

    printf("Third number: %d\n", numbers[2]);
    printf("Greeting: %s", greeting);

    return 0;
}
```

In this example, we define an integer array **numbers** and a character array **greeting**. We then access specific elements of the arrays and print them.

6. Pointers and Memory Management

Description: Pointers in C are variables that store memory addresses. They provide a powerful way to manipulate memory and data, enabling functionalities like dynamic memory allocation and efficient passing of large data structures to functions.

Understanding pointers is crucial for advanced memory management and efficient data handling. Pointers facilitate direct access to memory addresses, allowing for optimizations and flexibility in data manipulation.

Example:

```
#include <stdio.h>


int main() {
    int num = 10;
    int *ptr = &num;  // Pointer storing the address of num


    printf("Value of num: %d\n", num);
    printf("Address of num: %p\n", (void*)&num);
    printf("Value at the address stored in ptr: %d", *ptr);


    return 0;
```

}In this example, we declare a pointer **ptr** that stores the address of the variable **num**. We then print the value of **num**, the address of **num**, and the value at the address stored in **ptr**.

7. File Handling

Description: File handling in C involves reading from and writing to files. It's a crucial aspect for programs that need to store or retrieve data from external sources.

Understanding file handling is essential for creating applications that deal with persistent data storage and retrieval. Files are essential for long-term data storage, and file handling operations enable reading from and writing to files.

Example:

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

```
int main() {
    FILE *file;
    char data[100] = "This is a sample text.";

    file = fopen("sample.txt", "w");
    if (file == NULL) {
        printf("Error opening file.");
        return 1;
    }

    fprintf(file, "%s", data);
    fclose(file);

    return 0;
```

}In this example, we write data to a file named "sample.txt".

8. Structures and Unions

Description: Structures and unions allow you to create custom data types in C by grouping different types of data under a single name. Structures are collections of variables under one name, each called a member. Unions, on the other hand, allow storing only one value out of all its members at a time.

Understanding structures and unions is fundamental for organizing complex data and creating custom data types that efficiently represent real-world entities.

Example:

```
#include <stdio.h>

struct Point {
    int x;
    int y;
};


int main() {
    struct Point p1 = {5, 10};


    printf("Coordinates of Point: (%d, %d)", p1.x, p1.y);


    return 0;
```

}In this example, we define a structure **Point** with two integer members **x** and **y**. We then create a variable of type **Point** and print its coordinates.

9. Dynamic Memory Allocation

Description: Dynamic memory allocation in C allows you to allocate memory during program execution. It provides flexibility and is essential when the size of data is not known beforehand.

Understanding dynamic memory allocation is crucial for managing memory efficiently, especially for variable-sized data structures.

Example:

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


int main() {
    int *ptr;
    ptr = (int *)malloc(5 * sizeof(int));


    if (ptr == NULL) {
        printf("Memory allocation failed.");
        return 1;
    }

    for (int i = 0; i < 5; i++) {
        ptr[i] = i + 1;
    }

    printf("Dynamic Array: ");
    for (int i = 0; i < 5; i++) {
        printf("%d ", ptr[i]);
    }

    free(ptr);
    return 0;
```

}In this example, we dynamically allocate memory for an integer array of size 5.

10. Preprocessor Directives and Macros

Description: Preprocessor directives are commands that are executed before the compilation of the program. They are used for various tasks such as including header files, defining macros, and conditionally compiling code.

Understanding preprocessor directives and macros is essential for creating efficient and maintainable code.

Example:

```
#include <stdio.h>
#define PI 3.14159

int main() {
    double radius = 5.0;
    double area = PI * radius * radius;
    printf("Area of the circle: %lf", area);


    return 0;
}
```

In this example, we define a macro **PI** and use it to calculate the area of a circle.