VISVESVARAYA TECHNOLOGICAL UNIVERSITY

Jnana Sangama, Belagavi-590018

Mini Project Report on C Language

"TO-DO LIST MANAGER"

SUBMITTED BY: CHANDANA M C

UNDER THE GUIDENCE OF

Dr CHETHAN K C

ASSISTANT PROFESSOR

Dept. of CSE

GEC, RAMANAGARA

DEPARTMENT OF COMPUTER SCIENCE ENGINEERING

GOVERNMENT ENGINEERING COLLEGE RAMANAGARA-562159

GOVERNMENT ENGINEERING COLLEGE

Affiliated to VTU Belagavi, Recognized

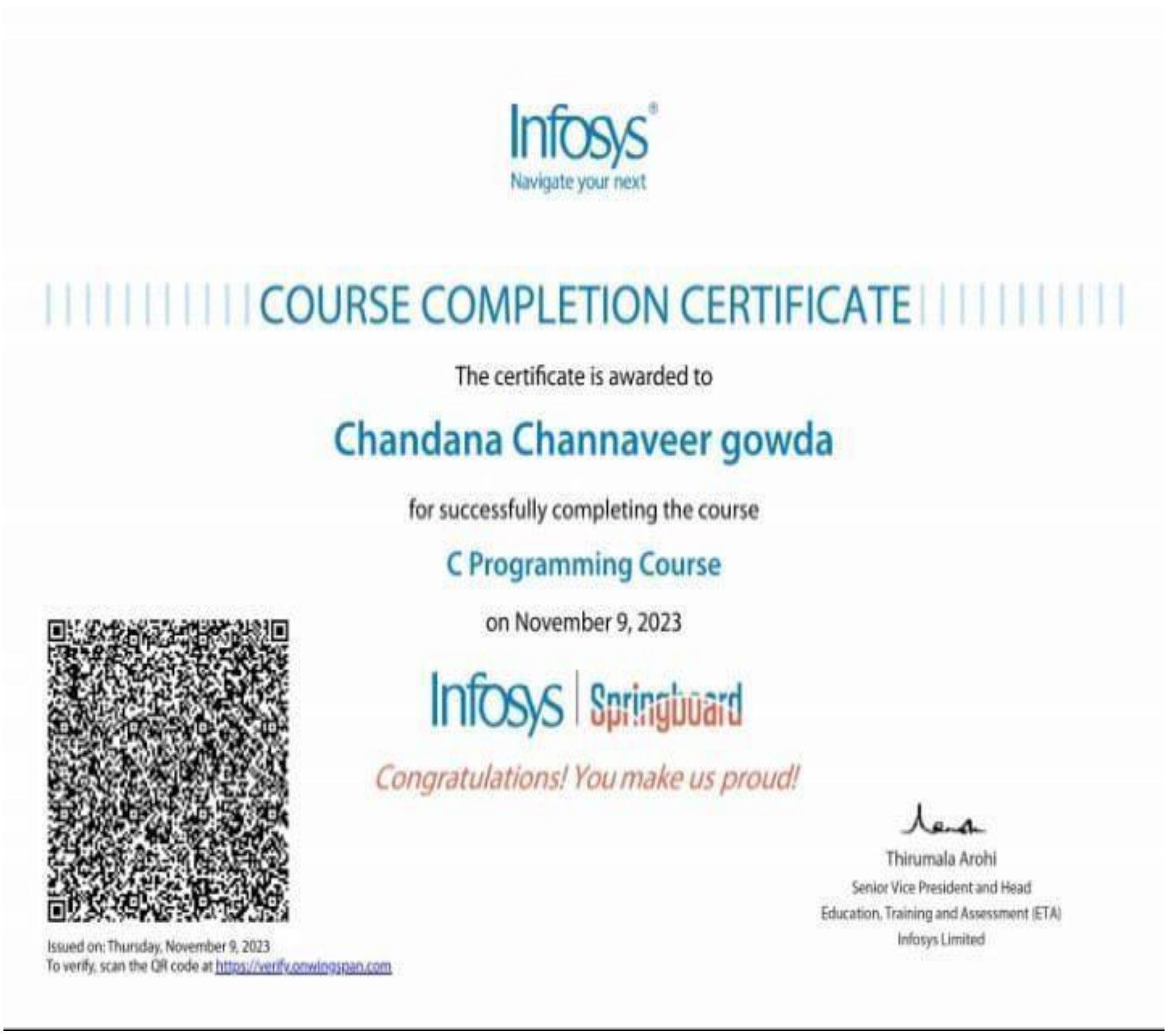
AICTE-New Delhi BM Road,

Ramanagara-562 159

CERTIFICATE

Certified that CHANDANA M C has submitted mini project report on C language carried out during First semester, as part of internal assessment for the subject PRINCIPLE OF PROGRAMMING USING C(BPOPS103)

(Signature of the guide)



Infosys
Navigate your next

COURSE COMPLETION CERTIFICATE

The certificate is awarded to

Chandana Channaveer gowda

for successfully completing the course

C Programming Course

on November 9, 2023

Infosys | Springboard

Congratulations! You make us proud!

[illegible]
Thirumala Arohi
Senior Vice President and Head
Education, Training and Assessment (ETA)
Infosys Limited

Issued on: Thursday, November 9, 2023
To verify, scan the QR code at https://verify.onwingspan.com

TABLE CONTENTS

ABSTRACTS

This C program generates a textual calendar for a specified year, incorporating leap year calculations and day code algorithms. It prompts the user to input a year, determines whether it is a leap year, dynamically adjusts the number of days in February accordingly, and prints a formatted calendar for each month of the given year. The modular structure and functions enhance code readability and maintainability, providing a simple yet effective console-based calendar generation tool.

CHAPTER 1

INTRODUCTION

The presented C program serves the purpose of generating a textual calendar for a specified year. This program incorporates essential concepts of C programming, including conditionals, loops, arrays, and modular functions. It dynamically adjusts the number of days in February based on leap year calculations and employs a calculated day code to determine the starting day of the week for January 1st.

The user is prompted to input a year through the console, and the program intelligently adjusts for leap years, ensuring accurate calendar generation. The modular structure enhances code organization and readability, making it suitable for educational purposes, especially for those learning C programming.

This program not only provides a practical tool for visualizing calendars but also serves as an educational example, showcasing the implementation of logical calculations and user interaction in a console environment. The following sections will delve into the specific functionalities, structure, and objectives of this program.

C programming code is a general purpose, computer programming language. It supports structured programming, variable scope and recursion. While a static type system prevents many unintended operations. By design programming C, C provides constructs that map efficiently to typical machine instruction. Therefore it has found lasting use in applications that had formerly been coded in assembly language include operating systems as well as various application software for computers ranging from supercomputers to embedded systems

The #include is a "preprocessor" directive that tells the compiler to put code from the header called stdio.h into our program before actually creating the executable. By including header files, you can gain access to many different functions-- the print functions are included in stdio.h. The next important line is int main(). This line tells the compiler that there is a function named main, and that the function returns an integer, hence int. If you don't want a return value then use void. The "curly braces" ((and)) signal the beginning and end of functions and other code blocks. The print function is the standard C way of displag outyinput on the screen. The quotes tell the compiler that you want to output the literal string as-is (almost). The "n' sequence is actually treated as a single character that stands for a newline. C has three types to choose among alternative courses of action: if, if, else, and switch. While, for, do-while are specifies a block of one or more statements that are repeatedly executed until a condition is satisfied.

Finally, at the end of the program, we return a value from main to the operating system by using the return statement. This return value is important as it can be used to tell the operating system whether our program succeeded or not. A return value of 0 means success.

CHAPTER 02

IMPORTANCE OF TO DO LIST MANAGER

1. Purpose:

- The primary purpose of a To-Do List Manager is to assist users in maintaining a clear overview of their tasks and responsibilities.
- It acts as a digital or physical reminder system, aiding in time management and productivity.

2. Features:

- Task Entry:Users can input tasks into the manager, often including details such as task names, due dates, and additional notes.
- Task Display: The manager provides a user-friendly display of tasks, allowing users to easily view and comprehend their to-do items.
- Priority Setting: Users can assign priorities to tasks, helping them focus on high-priority items first.
- Task Completion: Tasks can be marked as completed, providing a sense of accomplishment and aiding in progress tracking.
- Task Removal: Users can remove or archive completed or unnecessary tasks to keep the list uncluttered.

3. Benefits:

- Organization: A To-Do List Manager promotes organization by centralizing all tasks in one location, preventing information overload.
- Productivity: It enhances productivity by helping users prioritize tasks, set goals, and manage time effectively.
- Reduced Stress: The visual representation of tasks reduces stress by offering a clear

roadmap for completing them.

4. Implementation:

- To-Do List Managers can be implemented through various mediums, including mobile apps, web applications, desktop software, or even traditional pen-and-paper formats.
- Digital implementations often offer additional features such as synchronization across devices, reminders, and collaboration options.

5. Customization:

- To cater to diverse user preferences, To-Do List Managers often provide customization options, allowing users to tailor the tool to their specific needs.

6. Continuous Improvement:

- The dynamic nature of tasks requires To-Do List Managers to be adaptable. Users may need to update, reprioritize, or add new tasks regularly.

In summary, a To-Do List Manager is a versatile tool designed to simplify task management, improve productivity, and reduce the mental load associated with remembering numerous responsibilities. Its effectiveness lies in its simplicity and user-centric design, making it a valuable companion for individuals seeking to enhance their time management skills.

7. Readability and Maintainability:

- Incorporates meaningful variable names and comments to enhance code clarity.
- Uses arrays for month names and days in each month to improve code organization.

8. User-Friendly Output:

- Presents the calendar in a clear and user-friendly format, making it easy for users to visualize the month-by-month layout.

9. Simple Input Handling:

- Expects a single input from the user, simplifying the interaction process.

10. Educational Purpose:

- The program serves as an educational example for learning C programming, featuring common concepts such as conditionals, loops, arrays, and modular functions.

CHAPTER 03

COMPLETE SOURCE CODE

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

#define MAX_TASKS 10
#define MAX_TASK_LENGTH 50

struct ToDoList {
    char tasks[MAX_TASKS][MAX_TASK_LENGTH];
    int taskCount;
};

void initializeList(struct ToDoList *list) {
    list->taskCount = 0;
}

void addTask(struct ToDoList *list, const char *task) {
    if (list->taskCount < MAX_TASKS) {
        strcpy(list->tasks[list->taskCount], task);
        list->taskCount++;
        printf("Task added successfully.\n");
    } else {
        printf("Task list is full. Cannot add more tasks.\n");
    }
}

void displayTasks(const struct ToDoList *list) {
    if (list->taskCount > 0) {
        printf("Tasks:\n");
        for (int i = 0; i < list->taskCount; i++) {
            printf("%d. %s\n", i + 1, list->tasks[i]);
        }
    } else { printf("No tasks available.\n");

}
}

void removeTask(struct ToDoList *list, int taskIndex) {
    if (taskIndex >= 1 && taskIndex <= list->taskCount) {
        for (int i = taskIndex - 1; i < list->taskCount - 1; i++) {
            strcpy(list->tasks[i], list->tasks[i + 1]);
        }
        list->taskCount--;
        printf("Task removed successfully.\n");
    } else {
        printf("Invalid task index.\n");
    }
}int main() {
    struct ToDoList myList;
    initializeList(&myList);

    char task[MAX_TASK_LENGTH];


    do {
        printf("\n1. Add Task\n2. Display Tasks\n3. Remove Task\n4. Quit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter task: ");
                scanf(" %[^\n]", task);
                addTask(&myList, task);
                break;
            case 2:
                displayTasks(&myList);
                break;



case 3:
                printf("Enter task index to remove: ");
                int indexToRemove;
                scanf("%d", &indexToRemove);
                removeTask(&myList, indexToRemove);
                break;
            case 4:
                printf("Exiting program. Goodbye!\n");
                break;
            default:
                printf("Invalid choice. Please enter a valid option.\n");
        }

    } while (choice != 4);

    return 0;
}
```

CHAPTER 04

OUTPUT

```
1. Add Task
2. Display Tasks
3. Remove Task
4. Quit
Enter your choice: 1
Enter task: Read
Task added successfully.

1. Add Task
2. Display Tasks
3. Remove Task
4. Quit
Enter your choice: 2
Tasks:
1. Read

1. Add Task
2. Display Tasks
3. Remove Task
4. Quit
Enter your choice: 3
Enter task index to remove: 1
Task removed successfully.

1. Add Task
2. Display Tasks
3. Remove Task
4. Quit
Enter your choice: 4
Exiting
 program.

Goodbye!
```

CONCLUSION

1. Organization and Clarity:

- The To-Do List Manager provides a structured framework for organizing tasks, offering users a clear and concise view of their responsibilities.

2. Effective Time Management:

- By enabling prioritization and time allocation, the manager empowers users to make the most of their time, focusing on tasks that align with their goals and objectives.

3. Stress Reduction and Mental Well-being:

- Offloading tasks onto a list reduces the mental load associated with trying to remember everything, leading to decreased stress and improved mental well-being.

4. Motivation and Goal Alignment:

- Regularly checking off completed tasks instills a sense of accomplishment, motivating users to persist in their efforts. The manager helps align daily activities with broader goals.

5. Adaptability and Flexibility:

- To-Do List Managers are dynamic tools that allow users to adapt and reprioritize tasks based on changing circumstances, fostering adaptability and flexibility.

6. Collaboration and Accountability:

- In professional settings, these managers facilitate collaboration by allowing teams to share tasks and responsibilities. Users are held accountable for their tasks through the transparent tracking provided by the manager.

7. Consistency and Routine:

- Regular use of a To-Do List Manager establishes consistency in task management, encouraging users to create routines for planning and executing their daily activities.

8. Enhanced Productivity:

- The manager aids in maintaining focus on current tasks, avoiding overwhelm and boosting overall productivity. Users can efficiently track progress and identify areas for improvement.

9. Encouragement of Continuous Improvement:

- To-Do List Managers encourage a continuous improvement mindset. Users can refine their task management processes, identify patterns, and optimize their workflows over time.

10. Digital and Traditional Formats:

- To-Do List Managers are versatile, available in both digital and traditional formats. Users can choose the medium that best suits their preferences and workflow.