ST. XAVIER'S COLLEGE

MAITIGHAR, KATHMANDU, NEPAL
Phone: 01-5321365, 01-5344636
Email: ktm@sxc.edu.np

LAB ASSIGNMENT NUMBER: 11

"C PROGRAMMING"

Submitted By	Submitted To	Signature
Name: Shreyashkar Shah Roll No: 1025 Class: 11 Section: J	Mr. Jaya Sundar Shilpakar Department of Computer Science, St. Xavier's College	

Submission Date: 10th January, 2025

TABLE OF CONTENTS

SOFTWARE DEVELOPMENT

Software development involves a structured process to create, test, and deploy software solutions. Key stages include planning, analysis, design, development, testing, deployment, and maintenance. Various methodologies like Waterfall, Agile, Scrum, and DevOps guide this process. Essential skills for developers include problem-solving, critical thinking, creativity, communication, teamwork, and continuous learning. Successful software development focuses on security, user experience, performance, and scalability, ensuring reliable and efficient solutions.

SOFTWARE ENGINEERING

Software engineering is a systematic approach to designing, developing, testing, and deploying software applications. It involves applying engineering principles to software development to ensure quality, efficiency, and maintainability. Key aspects of software engineering include requirements gathering, system design, coding, testing, debugging, and deployment. Software engineers utilize programming languages, software development tools, and methodologies like Agile and Waterfall to build robust and scalable software solutions.

PROGRAMMING

Software engineering is a systematic approach to designing, developing, testing, and deploying software applications. It involves applying engineering principles to software development to ensure quality, efficiency, and maintainability. Key aspects of software engineering include requirements gathering, system design, coding, testing, debugging, and deployment. Software engineers utilize programming languages, software development tools, and methodologies like Agile and Waterfall to build robust and scalable software solutions.

CODING

Coding, the art of instructing computers, involves writing instructions in specific languages that machines can understand. It requires logical thinking, problem-solving skills, and creativity. Programmers break down complex problems into smaller, manageable steps, then translate these steps into code using languages like Python, Java, C++, and JavaScript. Coding is essential for developing software, websites, apps, and countless other digital products, driving innovation and shaping the modern world.

SOFTWARE DEVELOPMENT LIFE CYCLE

Here are the key points on the Software Development Life Cycle (SDLC), detailing each phase:

1) **Planning:** This initial phase involves defining the project scope, objectives, and feasibility. It includes resource allocation, budgeting, and timeline estimation to ensure that the project is viable and aligns with organizational goals.
2) **Requirements Gathering and Analysis:** In this phase, developers collect detailed functional and non-functional requirements from stakeholders. This information is analyzed and documented to provide a clear understanding of what the software needs to achieve.
3) **Design:** The design phase focuses on creating the architecture of the software. This includes high-level design, which defines system architecture, and low-level design, specifying detailed components and interactions. It serves as a blueprint for developers.
4) **Implementation (Coding):** During this phase, developers write the code according to the design specifications. It often involves multiple coding practices and preliminary unit testing to ensure that individual components function correctly.
5) **Testing:** In the testing phase, the software undergoes rigorous validation to ensure it meets the specified requirements. Various testing methods, such as integration testing, system testing, and acceptance testing, are used to identify and fix defects.
6) **Deployment:** This phase involves releasing the software to a live environment. It includes installation, configuration, user training, and documentation to ensure that the end-users can effectively use the software.
7) **Maintenance:** The final phase encompasses ongoing support and maintenance of the software. It involves fixing any issues that arise, implementing updates, and adding new features based on user feedback and changing requirements.

SYSTEM STUDY (FEASIBILITY STUDY)

System study is a crucial phase in the development of an information system, involving a comprehensive analysis of the current system and its requirements. It begins with gathering detailed information about existing processes, workflows, and user needs through various techniques like interviews, surveys, and observations. This phase evaluates the strengths and weaknesses of the current system, identifying inefficiencies and areas for improvement. The findings are then documented in a systematic manner, outlining requirements for the new system. Key components of a system study include problem identification, feasibility analysis, and requirement specification. Problem identification focuses on understanding what issues the current system faces, while feasibility analysis assesses the technical, operational, and economic viability of potential solutions. Requirement specification clearly defines what the new system must achieve to meet user needs and enhance productivity. The outcomes of a system study serve as a foundational reference for subsequent phases, ensuring that the new system aligns with organizational goals and user expectations. Overall, a thorough system study is vital to developing effective and efficient information systems that deliver significant value to users and stakeholders.

SYSTEM ANALYSIS (REQUIREMENT GATHERING)

System analysis is a critical phase in software development, focusing on understanding the existing system and identifying areas for improvement. It involves gathering detailed information about the system's current processes, data flows, and user requirements. Analysts use various techniques like interviews, surveys, and observation to collect data. The collected data is then analyzed to identify inefficiencies, bottlenecks, and potential problems. By understanding the system's strengths and weaknesses, analysts can propose solutions and recommendations for improvement. This analysis forms the foundation for the design and development phases, ensuring that the new system aligns with the organization's needs and goals.

SYSTEM DESIGN

System design involves creating a blueprint for a system's architecture, components, and interfaces. It focuses on defining the system's structure, functionality, and how its various parts interact. Key considerations in system design include scalability, performance, security, usability, and maintainability. The process typically involves understanding the system's requirements, identifying key components, designing the system's architecture, defining data flow and control flow, and considering potential failure scenarios. Effective system design ensures that the system meets its intended goals, is efficient, and can adapt to future changes.

SYSTEM DEVELOPMENT

System design involves creating a blueprint for a system's architecture, components, and interfaces. It focuses on defining the system's structure, functionality, and how its various parts interact. Key considerations in system design include scalability, performance, security, usability, and maintainability. The process typically involves understanding the system's requirements, identifying key components, designing the system's architecture, defining data flow and control flow, and considering potential failure scenarios. Effective system design ensures that the system meets its intended goals, is efficient, and can adapt to future changes.

SYSTEM TESTING

System testing evaluates the overall functionality and performance of an integrated software system. It ensures the system meets specified requirements and is suitable for delivery. Key types of system testing include:

1) **Functional Testing:** Verifies that the system performs its intended functions correctly.
2) **Non-Functional Testing:** Assesses non-functional aspects like performance, security, usability, and reliability.
3) **Performance Testing:** Evaluates the system's response time, scalability, and resource utilization under various load conditions.
4) **Security Testing:** Identifies vulnerabilities and weaknesses to protect the system from attacks.
5) **Usability Testing:** Assesses the system's ease of use and user experience.
6) **Stress Testing:** Determines the system's breaking point by pushing it beyond its expected load.
7) **Compatibility Testing:** Ensures the system functions correctly across different hardware, software, and network configurations.

SYSTEM IMPLEMENTATION

System implementation is the process of introducing a new system into an organization. It involves several key steps, including planning and preparation, system design and development, testing and quality assurance, user training and education, data migration and conversion, system deployment and cutover, and post-implementation support and evaluation. Effective system implementation requires careful planning, strong project management, and active user involvement to ensure a smooth transition and successful adoption of the new system.

SYSTEM MAINTENANCE

System maintenance is the ongoing process of keeping software and hardware systems operational, secure, and efficient. It involves a range of activities, including regular updates and patches to address security vulnerabilities and improve performance, bug fixes to resolve issues and errors, and feature enhancements to add new functionalities or improve existing ones. Additionally, system maintenance includes routine checks, such as backups and data recovery procedures, to protect against data loss. By prioritizing system maintenance, organizations can ensure the reliability, scalability, and overall health of their IT infrastructure.

TASKS

1. **Data Types:**
   Data types specify the type of data that can be stored in a variable. Common data types in C are `int`, `float`, `double`, and `char`.
   Codes:

```c
#include <stdio.h>

int main() {

    int num = 25;
    float f = 3.14;
    double d = 123.456;
    char ch = 'A';

    printf("Integer: %d\n", num);
    printf("Float: %.2f\n", f);
    printf("Double: %.3f\n", d);
    printf("Character: %c\n", ch);

    return 0;
}
```

   Output:

```
C:\Users\STUDENTS.DESKTOP-E64T97K\Desktop\Untitled1.exe
Integer: 25
Float: 3.14
Double: 123.456
Character: A

--------------------------------
Process exited after 0.006308 seconds with return value 0
Press any key to continue . . .
```

2. **Variables**
   Variables are used to store values that can change during the program execution. You must declare a variable before using it.
   Codes:

```c
#include <stdio.h>

int main() {
    int num = 10;
    float f = 5.5;
    char ch = 'B';

    // Using variables
    printf("Integer value: %d\n", num);
    printf("Float value: %.2f\n", f);
    printf("Character value: %c\n", ch);

    return 0;
}

```

Output:

```
C:\Users\STUDENTS.DESKTOP-E64T97K\Desktop\Untitled1.exe
Integer value: 10
Float value: 5.50
Character value: B

--------------------------------
Process exited after 0.03018 seconds with return value 0
Press any key to continue . . .
```

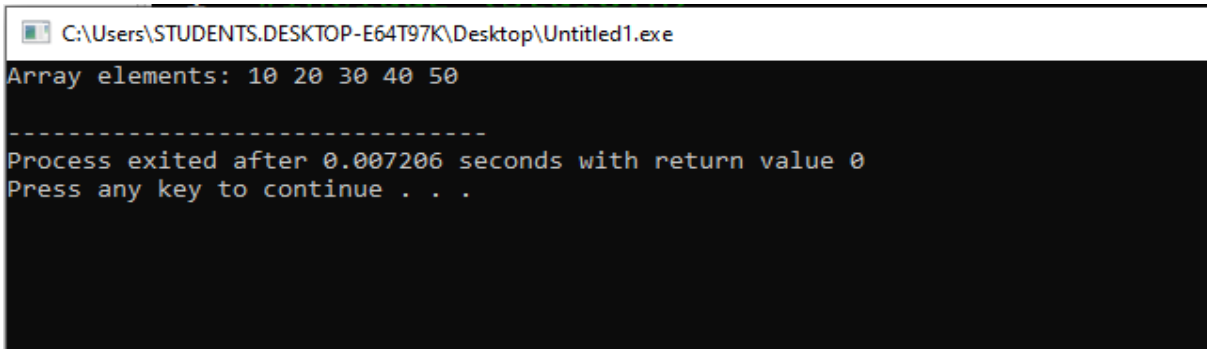
3. **Arrays**

An array is a collection of variables that are of the same type. The size of the array is fixed.

Code:

```c
#include <stdio.h>

int main() {

    int arr[] = {10, 20, 30, 40, 50};
    int size = sizeof(arr) / sizeof(arr[0]);


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

    return 0;
}

```

Output:



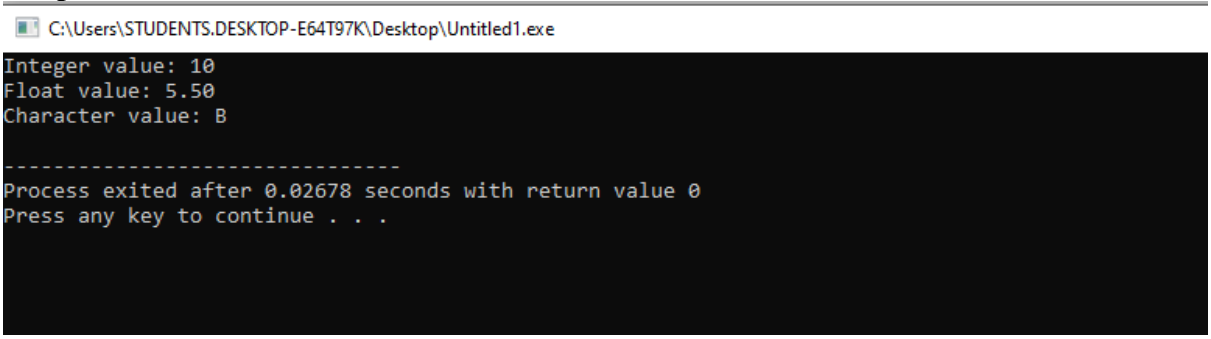
4. **Strings:**

Strings in C are arrays of characters terminated by a null character (`\0`).

Code:

```c
#include <stdio.h>

int main() {
    int num = 10;
    float f = 5.5;
    char ch = 'B';


    printf("Integer value: %d\n", num);
    printf("Float value: %.2f\n", f);
    printf("Character value: %c\n", ch);

    return 0;
}

```

Output:



5. **Operators:**

Operators are used to perform operations on variables and values. Examples of operators are:

1. **Arithmetic**: `+`, `-`, `*`, `/`, `%`
2. **Relational**: `==`, `!=`, `>`, `<`, `>=`, `<=`
3. **Logical**: `&&`, `||`, `!`

Codes:

Users > shreyashkar > Desktop > C Programming > C abc.c > ...

```c
#include <stdio.h>

int main() {
    int a = 10, b = 5;

    printf("Arithmetic Operators:\n");
    printf("a + b = %d\n", a + b);
    printf("a - b = %d\n", a - b);
    printf("a * b = %d\n", a * b);
    printf("a / b = %d\n", a / b);

    printf("\nRelational Operators:\n");
    printf("a > b: %d\n", a > b);
    printf("a < b: %d\n", a < b);

    printf("\nLogical Operators:\n");
    printf("(a > b) && (a != 0): %d\n", (a > b) && (a != 0));
    printf("(a < b) || (b == 5): %d\n", (a < b) || (b == 5));

    printf("\nBitwise Operators:\n");
    printf("a & b = %d\n", a & b);
    printf("a | b = %d\n", a | b);
    printf("a ^ b = %d\n", a ^ b);

    return 0;
}
```

Output:

```
Arithmetic Operators:
a + b = 15
a - b = 5
a * b = 50
a / b = 2

Relational Operators:
a > b: 1
a < b: 0

Logical Operators:
(a > b) && (a != 0): 1
(a < b) || (b == 5): 1

Bitwise Operators:
a & b = 0
a | b = 15
a ^ b = 15
```

6. **Functions:**
   A function is a block of code that performs a specific task and can be reused multiple times in a program.
   Codes:

```c
#include <stdio.h>

int add(int, int);
int subtract(int, int);

int main() {
    int x = 20, y = 10;

    printf("Using Functions:\n");
    printf("Addition: %d\n", add(x, y));
    printf("Subtraction: %d\n", subtract(x, y));

    return 0;
}

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

int subtract(int a, int b) {
    return a - b;
}
```

Output:

```
Using Functions:
Addition: 30
Subtraction: 10
```

7. **Escape Sequence Characters**

An escape sequence in C is a combination of characters that starts with a backslash (\) followed by one or more characters.

Codes:

```c
#include <stdio.h>

int main() {
    printf("Escape Sequence Characters:\n");
    printf("New Line: Hello\nWorld\n");
    printf("Tab: Hello\tWorld\n");
    printf("Double Quotes: \"Hello World\"\n");
    printf("Backslash: Hello\\World\n");

    return 0;
}
```

Output:

```
Escape Sequence Characters:
New Line: Hello
World
Tab: Hello      World
Double Quotes: "Hello World"
Backslash: Hello\World
```

CONCLUSION

This lab has provided valuable insights into both the theoretical and practical aspects of software development. Through the exploration of the Software Development Life Cycle (SDLC), I gained a comprehensive understanding of the phases involved, such as system analysis, design, development, testing, and maintenance. Each phase plays a critical role in ensuring the success of a software project, and I now

recognize the importance of systematic planning, requirement gathering, and feasibility studies in delivering high-quality software solutions. Additionally, the design process, using tools like pseudo code, algorithms, and flowcharts, allows for effective communication and problem-solving before actual development begins.

The C programming tasks further strengthened my knowledge of fundamental programming concepts. I learned to work with different data types, variables, arrays, and strings, which are essential for handling and processing information efficiently. Understanding operators, functions, and escape sequences enriched my ability to write clean, functional, and well-organized code. Overall, this lab has enhanced my understanding of software engineering principles while providing practical experience in programming, which will be invaluable in future software development projects.