



SGCP's

Guru Nanak Khalsa College
of Arts, Science & Commerce (Autonomous)

NAME : MS. PRARTHI HRISHIT KOTHARI

CLASS : M. Sc. PART I

COURSE : BIOINFORMATICS

ACADEMIC YEAR : 2023 – 2024

ROLL NO. : 115

PAPER CODE : GNKPSBI2501

**COURSE TITLE : INTRODUCTION TO PROGRAMMING
LANGUAGES AND DATABASES**



SGCP's

Guru Nanak Khalsa College
of Arts, Science & Commerce (Autonomous)

DEPARTMENT OF BIOINFORMATICS

CERTIFICATE

This is to certify that Ms. Prarthi Hrishit Kothari (Roll No. – 115) of M. Sc. Part I Bioinformatics has satisfactorily completed the practical Semester I course prescribed by the University of Mumbai during the academic year 2023 – 2024.

Teacher – in – Charge
(Signature)

Head of Department
(Signature)

External Examiner
(Signature)

INDEX

Practical No.	Experiment / Practical Name	Page No.	Date	Signature
1.	SQL : To collect information and store as relational database using SQL commands.	01	30/08/2023	
2.	C++ : To perform basic input and output operations using the header file in C++ programming language.	17	13/10/2023	
3.	C++ : To demonstrate the use of conditional statements and loops in C++ programming language.	26	16/10/2023	
4.	C++ : To demonstrate the use of arrays in C++ programming language.	37	02/11/2023	
5.	C++ : To demonstrate the use of functions and structures in C++ programming language.	48	02/11/2023	
6.	C++ : To demonstrate the use of string manipulation in C++ programming language.	55	02/11/2023	
7.	C++ : To explore and demonstrate the concept of classes, objects and encapsulation in C++ programming language.	62	03/11/2023	
8.	C++ : To explore and demonstrate the concept of inheritance in C++ programming language.	66	03/11/2023	
9.	C++ : To explore and demonstrate the concepts of polymorphism, virtual function and friend function in C++ programming language.	73	06/11/2023	
10.	C++ : To explore and demonstrate the concept of constructors and destructors in C++ programming language.	79	06/11/2023	

Date – 30/08/2023

Practical 1

SQL : Structure Query Language

AIM :

To collect information and store as relational database using SQL commands.

INTRODUCTION :

SQL stands for Structured Query Language. It is a computer language used for storing and managing, manipulating and retrieving data in relational database management system (RDBMS). It is a standard language for Relational Database System. It enables a user to create, read, update and delete relational databases and tables.

All the RDBMS like MySQL, Informix, Oracle, MS Access and SQL Server use SQL as their standard database language. SQL can be integrated with other programming languages, for instance, embedding SQL queries with the Java programming language to build high – performing data processing applications.

SQL standards are a set of formally defined guidelines of the Structured Query Language (SQL). The American National Standards Institute (ANSI) and the International Organization of Standardization (ISO) adopted the SQL standards in 1986.

Components of an SQL system include –

- 1. SQL Table** – It is the basic element of a relational database which consists of rows and columns representing different entities, their data attributes and the various relationships between the data values.
- 2. SQL Statements** – Also known as SQL Queries, these statements are valid instructions that relational database management system understand. The SQL Language elements that constitute a correct SQL statement include – operators, identifiers, variables and their data types, and search conditions.
- 3. Stored Procedures** – They are a collection of one or more SQL statements stored in the relational database used to improve the efficiency and performance of the overall SQL Relational Database System.

Advantages of SQL –

1. **Faster and Efficient Query Processing** – SQL can quickly and efficiently retrieve a large volume of data records from a database. Compared to an unstructured database such as MongoDB, it is a relational database that can characterize the data in a structured way.
Operations like – insertion, deletion, querying, manipulation and calculations – on data via analytical queries in a relational database can be accomplished in a matter of seconds.
2. **Portable** – SQL is highly portable because it is employed in programs on PCs, servers, tablets and independent laptops running operating systems such as Windows, Linux, Mac and even some mobile phones. It can also be embedded with other programs based on the requirements.
3. **Standardized language** – It gives all users a consistent platform worldwide due to proper documentation, making it convenient for developers and businesses to access and shift to different databases.
4. **Data Integrity** – SQL ensures the consistency and accuracy of data by using constraints such as primary key, foreign key, NOT NULL and UNIQUE constraints.
Following are the 4 types of data integrity maintained by SQL –
 - (a) Entity Integrity – Ensures that there are no duplicate rows in a table.
 - (b) Domain Integrity – Enforces valid entries for a given column by restricting the data type.
 - (c) Referential Integrity – Rows cannot be deleted which are used by other records.
 - (d) User – defined Integrity – enforces some specific business rules.
5. **Flexibility** – SQL, being a versatile language, empowers users to perform complex data analysis and management tasks efficiently.

Disadvantages of SQL –

1. **Resource Intensive Scaling** – SQL databases typically scale up vertically by increasing hardware investment which is both costly and time – consuming.
2. **Partial control** – SQL does not provide programmers complete control over databases primarily due to hidden corporate rules.
3. **Cost inefficient** – some versions of SQL are expensive, making it inaccessible to programmers.
4. **Rigidity** – The schema of a SQL database must be specified before it can be used. They are rigid once installed, and changes are often complex and time – consuming, hence limiting the ability of developers and businesses to customize it according to their needs.
5. **Security threats** – Certain security threats / risks such as injection attacks may further compromise the integrity and confidentiality of data stored.

Data Types in SQL –

Category	Data Type	Syntax	Description
String	Character	CHAR	It stores string values containing any characters in a character set. It is defined to be of fixed length.
	Varying Character	VARCHAR VARCHAR2	It stores string values containing variable characters in a set of characters but of defined variable length.
	Binary Large Object	BLOB	It stores binary string values in hexadecimal format. It has a defined variable length.
Numeric	Numeric	NUMERIC NUMBER	It stores exact numbers with a defined precision and scale. It has decimals (similar to the datatype float in python).
	Integer	INT	It stores exact numbers with a pre – defined precision and scale to zero. It stores integer values.
Temporal	Timestamp	TIMESTAMP	It stores a moment an event occurs, using a definable fraction – of – a second precision.
	Timestamp with local	TIMESTAMP WITH LOCAL	It stores a moment an event occurs, using a definable fraction – of – a second precision, with respect to the local time zone.
	Timezone	TIMEZONE	It defines the timezone.
Boolean	Boolean	BOOLEAN	It checks the condition and stores the following values – True, False or Unknown.

SQL Commands –

Following are the 4 types of SQL Commands –

1. Data Definition Language (DDL) – Adding or deleting the table
2. Data Manipulation Language (DML) – Changing or inserting or replacing the data / table values
3. Data Control Language (DCL) – Giving a set of controls / rights / access to the group of users for a particular table
4. Data Query Language (DQL) – Stored data can be retrieved using a query and can be displayed / shown to the users.

Type of SQL Command	Command	Description	Syntax
Data Definition Language (DDL)	CREATE	It creates a database, new tables, a view of the table or other object in the database.	CREATE DATABASE <i>database_name</i> ; CREATE TABLE <i>table_name</i> (<i>column1_name datatype</i> , <i>column2_name datatype</i> , ...);
	ALTER	It modifies / alters an existing database object, such as the table.	ALTER TABLE <i>table_name</i> ADD <i>column_name datatype</i> ;
	DROP	It is deletes a database, an entire table, views of the table or other object in the database.	DROP DATABASE <i>database_name</i> ; DROP TABLE <i>table_name</i> ;
Data Manipulation Language (DML)	INSERT	It inserts a new record in a table.	INSERT INTO <i>table_name</i> (<i>column1_name</i> , <i>column2_name</i>) VALUES (<i>value1</i> , <i>value2</i>);
	UPDATE	It modifies an existing record.	UPDATE <i>table_name</i> SET <i>column_name</i> = <i>value</i> WHERE <i>condition</i> ;
	DELETE	It deletes an existing record in a table.	DELETE FROM <i>table_name</i> WHERE <i>condition</i> ;
	COMMIT	It is used to commit a transaction.	COMMIT;

	ROLLBACK	It is used to rollback a transaction in case any error occurs.	ROLLBACK;
Data Control Language (DCL)	GRANT	It assigns privileges to a user to access the database.	GRANT <i>priviledge_name</i> ON <i>database_name.table_name</i> TO <i>username</i> ;
	REVOKE	It used to take back the privileges granted to a user for a particular database.	REVOKE <i>priviledge_name</i> ON <i>database_name.table_name</i> FROM <i>username</i> ;
Data Query Language (DQL)	SELECT	It is used to retrieve certain records from one or more tables and display it in the console.	SELECT <i>column1_name</i> , <i>column2_name</i> ... FROM <i>table_name</i> ;
	SELECT DISTINCT	It is used to return only distinct (different) values of certain records from one or more tables and display it in the console.	SELECT DISTINCT <i>column1_name</i> , <i>column2_name</i> ... FROM <i>table_name</i> ;
	WHERE	It is a clause used to filter records based on a condition.	SELECT <i>column1_name</i> , <i>column2_name</i> ,... FROM <i>table_name</i> ;
	IN	It is an operator used to specify multiple values in a WHERE clause. It is used to retrieve only those records from a table that have the specified value.	SELECT <i>column1_name</i> , <i>column2_name</i> ,... FROM <i>table_name</i> WHERE <i>column_name</i> IN (<i>value1</i> , <i>value2</i>);
	NOT IN	It is an operator used to specify multiple values in a WHERE clause. It is used to retrieve records from a table except for	SELECT <i>column1_name</i> , <i>column2_name</i> ,... FROM <i>table_name</i> WHERE <i>column_name</i> NOT IN (<i>value1</i> , <i>value2</i>);

		those records that have the specified value.	
	IS NULL	It checks if the value stored in a record of a table is null.	SELECT <i>column1_name</i> , <i>column2_name</i> ,... FROM <i>table_name</i> WHERE <i>column_name</i> IS NULL;
	IS NOT NULL	It checks if the value stored in a record of a table is not null.	SELECT <i>column1_name</i> , <i>column2_name</i> ,... FROM <i>table_name</i> WHERE <i>column_name</i> IS NOT NULL;
	ORDER BY	It is used to sort the selected record set in either ascending (ASC) or descending (DESC) order.	SELECT <i>column1_name</i> , <i>column2_name</i> ,... FROM <i>table_name</i> WHERE <i>condition</i> ORDER BY <i>column_name</i> {ASC DESC};
	LIKE	It is an operator used in a WHERE clause used to search for a specified pattern in a column. Underscore (_) indicates a single character. Percent sign (%) indicates more than one characters.	SELECT <i>column1_name</i> , <i>column2_name</i> ,... FROM <i>table_name</i> WHERE <i>column_name</i> LIKE <i>pattern</i> ; -- SELECT * FROM CUSTOMERS WHERE CUSTOMER_NAME LIKE '_a%';
	GROUP BY	It is used with aggregate functions [COUNT(), MAX(), MIN(), SUM(), AVG()] to group the required result set by one or more columns.	SELECT <i>column1_name</i> , <i>column2_name</i> ,... FROM <i>table_name</i> WHERE <i>condition</i> GROUP BY <i>column_name</i> ;
	COUNT()	It returns the number of rows that matches a specified criterion.	SELECT COUNT (<i>column_name</i>) AS <i>column_name_to_be_displayed</i> FROM <i>table_name</i> WHERE <i>condition</i> ;

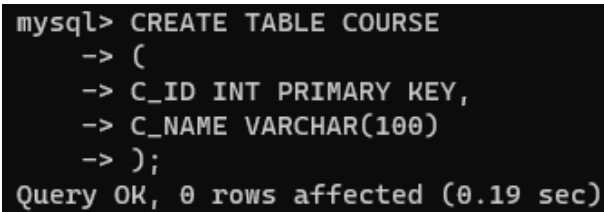
	MIN()	It returns the smallest value of the selected column.	SELECT MIN (<i>column_name</i>) AS <i>column_name_to_be_displayed</i> FROM <i>table_name</i> WHERE <i>condition</i> ;
	MAX()	It returns the largest value of the selected column.	SELECT MAX (<i>column_name</i>) AS <i>column_name_to_be_displayed</i> FROM <i>table_name</i> WHERE <i>condition</i> ;
	SUM()	It returns the total sum of a numeric column.	SELECT SUM (<i>column_name</i>) AS <i>column_name_to_be_displayed</i> FROM <i>table_name</i> WHERE <i>condition</i> ;
	AVG()	It returns the average value of a numeric column.	SELECT AVG (<i>column_name</i>) AS <i>column_name_to_be_displayed</i> FROM <i>table_name</i> WHERE <i>condition</i> ;

Question No. 1 – Create table COURSE and attributes are C_ID and C_NAME.

CODE :

```
mysql> CREATE TABLE COURSE
(
  C_ID INT PRIMARY KEY,
  C_NAME VARCHAR(100)
);
```

OUTPUT :



```
mysql> CREATE TABLE COURSE
-> (
-> C_ID INT PRIMARY KEY,
-> C_NAME VARCHAR(100)
-> );
Query OK, 0 rows affected (0.19 sec)
```

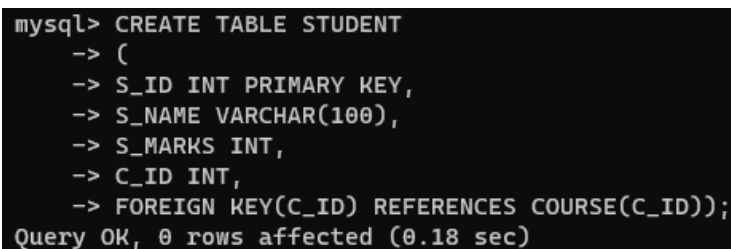
Figure 2 – Creating the table COURSE with the attributes C_ID (Primary Key) and C_NAME

Question No. 2 – Create table STUDENT and attributes are S_ID, S_NAME, S_MARKS and C_ID.

CODE :

```
mysql> CREATE TABLE STUDENT
(
  S_ID INT PRIMARY KEY,
  S_NAME VARCHAR (100),
  S_MARKS INT,
  C_ID INT,
  FOREIGN KEY(C_ID) REFERENCES COURSE(C_ID)
);
```

OUTPUT :



```
mysql> CREATE TABLE STUDENT
-> (
-> S_ID INT PRIMARY KEY,
-> S_NAME VARCHAR(100),
-> S_MARKS INT,
-> C_ID INT,
-> FOREIGN KEY(C_ID) REFERENCES COURSE(C_ID));
Query OK, 0 rows affected (0.18 sec)
```

Figure 3 – Creating the table STUDENT with the attributes S_ID (Primary Key), S_NAME, S_MARKS and C_ID (Foreign Key)

Question No. 3 – Insert values into table COURSE and STUDENT.

CODE 1 (for inserting values into table COURSE) :

```
mysql> INSERT INTO COURSE VALUES(101, 'Biology');  
mysql> INSERT INTO COURSE VALUES(102, 'Bioinformatics');  
mysql> INSERT INTO COURSE VALUES(103, 'Microbiology');
```

OUTPUT 1 (for table COURSE) :

```
mysql> SELECT * FROM COURSE;  
+-----+-----+  
| C_ID | C_NAME      |  
+-----+-----+  
| 101  | Biology     |  
| 102  | Bioinformatics |  
| 103  | Microbiology |  
+-----+-----+  
3 rows in set (0.00 sec)
```

Figure 4 – Inserting values in the table COURSE and displaying the table with the entered values

CODE 2 (for inserting values into table STUDENT) :

```
mysql> INSERT INTO STUDENT VALUES(1, 'Monika', 450, 101);  
mysql> INSERT INTO STUDENT VALUES(2, 'Niharika', 380, 101);  
mysql> INSERT INTO STUDENT VALUES(3, 'Vishal', 420, 101);  
mysql> INSERT INTO STUDENT VALUES(4, 'Amitabh', 480, 102);  
mysql> INSERT INTO STUDENT VALUES(5, 'Vivek', 360, 102);  
mysql> INSERT INTO STUDENT VALUES(6, 'Vipul', 450, 102);  
mysql> INSERT INTO STUDENT VALUES(7, 'Satish', 400, 103);  
mysql> INSERT INTO STUDENT VALUES(8, 'Geetika', 340, 103);
```

OUTPUT 2 (for table STUDENT) :

```
mysql> SELECT * FROM STUDENT;  
+-----+-----+-----+-----+  
| S_ID | S_NAME      | S_MARKS | C_ID |  
+-----+-----+-----+-----+  
| 1    | Monika      | 450     | 101  |  
| 2    | Niharika    | 380     | 101  |  
| 3    | Vishal      | 420     | 101  |  
| 4    | Amitabh     | 480     | 102  |  
| 5    | Vivek       | 360     | 102  |  
| 6    | Vipul       | 450     | 102  |  
| 7    | Satish      | 400     | 103  |  
| 8    | Geetika     | 340     | 103  |  
+-----+-----+-----+-----+  
8 rows in set (0.00 sec)
```

Figure 5 – Inserting values in the table STUDENT and displaying the table with the entered values

Question No. 4 – Write an SQL Query to fetch unique values of Course ID from STUDENT table.

CODE :

```
mysql> SELECT DISTINCT C_ID FROM STUDENT;
```

OUTPUT :

```
mysql> SELECT DISTINCT C_ID FROM STUDENT;
+-----+
| C_ID |
+-----+
| 101  |
| 102  |
| 103  |
+-----+
3 rows in set (0.00 sec)
```

Figure 6 – Fetching and displaying unique values of Course ID (C_ID) from the table STUDENT

Question No. 5 – Display the record of the student whose ID is 4.

CODE :

```
mysql> SELECT * FROM STUDENT WHERE S_ID = 4;
```

OUTPUT :

```
mysql> SELECT * FROM STUDENT WHERE S_ID = 4;
+-----+-----+-----+-----+
| S_ID | S_NAME | S_MARKS | C_ID |
+-----+-----+-----+-----+
| 4    | Amitabh | 480    | 102  |
+-----+-----+-----+-----+
1 row in set (0.00 sec)
```

Figure 7 – Fetching and displaying the record of the student from the table STUDENT whose Student ID (S_ID) is 4

Question No. 6 – Display the record of the student whose name starts with the letter ‘S’.

CODE :

```
mysql> SELECT * FROM STUDENT WHERE S_NAME LIKE 'S%';
```

OUTPUT :

```
mysql> SELECT * FROM STUDENT WHERE S_NAME LIKE 'S%';
+-----+-----+-----+-----+
| S_ID | S_NAME | S_MARKS | C_ID |
+-----+-----+-----+-----+
| 7 | Satish | 400 | 103 |
+-----+-----+-----+-----+
1 row in set (0.00 sec)
```

Figure 8 – Fetching and displaying the record of the student from the table STUDENT whose Name (S_NAME) starts with the letter ‘S’

Question No. 7 – Remove the record from table STUDENT whose student ID is 7.

CODE :

```
mysql> DELETE FROM STUDENT WHERE S_ID = 7;
```

OUTPUT :

```
mysql> DELETE FROM STUDENT WHERE S_ID = 7;
Query OK, 1 row affected (0.04 sec)

mysql> SELECT * FROM STUDENT;
+-----+-----+-----+-----+
| S_ID | S_NAME | S_MARKS | C_ID |
+-----+-----+-----+-----+
| 1 | Monika | 450 | 101 |
| 2 | Niharika | 380 | 101 |
| 3 | Vishal | 420 | 101 |
| 4 | Amitabh | 480 | 102 |
| 5 | Vivek | 360 | 102 |
| 6 | Vipul | 450 | 102 |
| 8 | Geetika | 340 | 103 |
+-----+-----+-----+-----+
7 rows in set (0.00 sec)
```

Figure 9 – Removing / Deleting the record of the student from the table STUDENT whose Student ID (S_ID) is 7

Question No. 8 – Add new column to table STUDENT as Address.

CODE :

```
mysql> ALTER TABLE STUDENT  
ADD S_ADDRESS VARCHAR(125) NOT NULL;
```

OUTPUT :

```
mysql> SELECT * FROM STUDENT;  
+-----+-----+-----+-----+-----+  
| S_ID | S_NAME | S_MARKS | C_ID | S_ADDRESS |  
+-----+-----+-----+-----+-----+  
| 1 | Monika | 450 | 101 | |  
| 2 | Niharika | 380 | 101 | |  
| 3 | Vishal | 420 | 101 | |  
| 4 | Amitabh | 480 | 102 | |  
| 5 | Vivek | 360 | 102 | |  
| 6 | Vipul | 450 | 102 | |  
| 8 | Geetika | 340 | 103 | |  
+-----+-----+-----+-----+-----+  
7 rows in set (0.00 sec)
```

Figure 10 – Adding a new column (S_ADDRESS) to the table STUDENT

Question No. 9 – Add values to the Address column.

CODE :

```
mysql> UPDATE STUDENT  
SET S_ADDRESS = 'MUMBAI'  
WHERE S_ID = 3;
```

OUTPUT :

```
mysql> SELECT * FROM STUDENT;  
+-----+-----+-----+-----+-----+  
| S_ID | S_NAME | S_MARKS | C_ID | S_ADDRESS |  
+-----+-----+-----+-----+-----+  
| 1 | Monika | 450 | 101 | |  
| 2 | Niharika | 380 | 101 | |  
| 3 | Vishal | 420 | 101 | MUMBAI |  
| 4 | Amitabh | 480 | 102 | |  
| 5 | Vivek | 360 | 102 | |  
| 6 | Vipul | 450 | 102 | |  
| 8 | Geetika | 340 | 103 | |  
+-----+-----+-----+-----+-----+  
7 rows in set (0.00 sec)
```

Figure 11 – Adding values to the Address (S_ADDRESS) column in the table STUDENT.

Question No. 10 – Display the record of student whose course ID is 101 and 102 using IN clause.

CODE :

```
mysql> SELECT * FROM STUDENT WHERE C_ID IN(101,102);
```

OUTPUT :

```
mysql> SELECT * FROM STUDENT WHERE C_ID IN(101,102);
+-----+-----+-----+-----+-----+
| S_ID | S_NAME | S_MARKS | C_ID | S_ADDRESS |
+-----+-----+-----+-----+-----+
| 1 | Monika | 450 | 101 | MUMBAI |
| 2 | Niharika | 380 | 101 | MUMBAI |
| 3 | Vishal | 420 | 101 | MUMBAI |
| 4 | Amitabh | 480 | 102 | MUMBAI |
| 5 | Vivek | 360 | 102 | MUMBAI |
| 6 | Vipul | 450 | 102 | MUMBAI |
+-----+-----+-----+-----+-----+
6 rows in set (0.00 sec)
```

Figure 12 – Fetching and displaying the record of the students from the table STUDENT whose Course ID (C_ID) is 101 and 102

Question No. 11 – Display the record of student by sorting in ascending order of marks column.

CODE :

```
mysql> SELECT * FROM STUDENT
      ORDER BY S_MARKS ASC;
```

OUTPUT :

```
+-----+-----+-----+-----+-----+
| S_ID | S_NAME | S_MARKS | C_ID | S_ADDRESS |
+-----+-----+-----+-----+-----+
| 8 | Geetika | 340 | 103 | MUMBAI |
| 5 | Vivek | 360 | 102 | MUMBAI |
| 2 | Niharika | 380 | 101 | MUMBAI |
| 3 | Vishal | 420 | 101 | MUMBAI |
| 1 | Monika | 450 | 101 | MUMBAI |
| 6 | Vipul | 450 | 102 | MUMBAI |
| 4 | Amitabh | 480 | 102 | MUMBAI |
+-----+-----+-----+-----+-----+
7 rows in set (0.00 sec)
```

Figure 13 – Fetching and displaying the record of the student from the table STUDENT whose Name (S_NAME) starts with the letter ‘S’

Question No. 12 – Calculate average, sum and max marks of student.

CODE :

```
mysql> SELECT AVG(S_MARKS), SUM(S_MARKS), MAX(S_MARKS) FROM STUDENT;
```

OUTPUT :

```
mysql> SELECT AVG(S_MARKS), SUM(S_MARKS), MAX(S_MARKS) FROM STUDENT;
+-----+-----+-----+
| AVG(S_MARKS) | SUM(S_MARKS) | MAX(S_MARKS) |
+-----+-----+-----+
|      411.4286 |          2880 |           480 |
+-----+-----+-----+
1 row in set (0.00 sec)
```

Figure 14 – Calculating and displaying the Average Marks, Total Sum of the Marks obtained and the Maximum Marks of the students from the table STUDENT

Question No. 13 – Calculate average, sum and max marks of student by department.

CODE :

```
mysql> SELECT AVG(S_MARKS), SUM(S_MARKS), MAX(S_MARKS), C_ID
FROM STUDENT
GROUP BY C_ID;
```

OUTPUT :

```
mysql> SELECT AVG(S_MARKS), SUM(S_MARKS), MAX(S_MARKS), C_ID FROM STUDENT
-> GROUP BY C_ID;
+-----+-----+-----+-----+
| AVG(S_MARKS) | SUM(S_MARKS) | MAX(S_MARKS) | C_ID |
+-----+-----+-----+-----+
|      416.6667 |          1250 |           450 |   101 |
|      430.0000 |          1290 |           480 |   102 |
|      340.0000 |           340 |           340 |   103 |
+-----+-----+-----+-----+
3 rows in set (0.00 sec)
```

Figure 15 – Calculating and displaying the Average Marks, Total Sum of the Marks obtained and the Maximum Marks of the students from the table STUDENT and further categorising (grouping) them by Course ID (C_ID)

RESULTS :

Using SQL (Structured Query Language) through the MySQL software as the Database Management System Software, a database was created and further used to create tables, termed as entities. Various attributes were assigned to the entities and values were inserted into the tables. Various commands were used to retrieve, display, delete, modify and arrange the data from the tables. Few commands were also used to perform arithmetic functions including calculating the average, total sum and maximum of the selected data.

CONCLUSION :

Using SQL (Structured Query Language) commands, information in the form of data was collected and stores as relational database and further studied via data retrieval, deletion, modification and display.

REFERENCES :

1. *SQL Tutorial*. (n.d.). <https://www.w3schools.com/sql/>
2. Groff, J., Weinberg, P., & Oppel, A. (2009, August 12). *SQL The Complete Reference, 3rd Edition*. McGraw-hill.
3. *Learn MySQL: A Comprehensive Guide to Using and Managing Databases*. (n.d.). <https://www.w3schools.in/mysql/>
4. *SQL Tutorial, Tutorials SQL*. (n.d.). <https://beginner-sql-tutorial.com/sql.htm>

Practical 2

C++ : Basic I/O Programs

AIM:

To perform basic input and output operations using the <iostream.h> header file in C++ programming language.

INTRODUCTION:

C++ is a high-level, general-purpose programming language designed for system and application programming. It was developed by Bjarne Stroustrup at Bell Labs in 1983 as an extension of the C programming language. C++ is an object-oriented, multi-paradigm language that supports procedural, functional, and generic programming styles.

One of the key features of C++ is its ability to support low-level, system-level programming, making it suitable for developing operating systems, device drivers, and other system software. At the same time, C++ also provides a rich set of libraries and features for high – level application programming, making it a popular choice for developing desktop applications, video games, and other complex applications.

ADVANTAGES OF C++

1. **Object – Oriented Programming** – The major principles of the Object – Oriented Programming language, C++ include classes, inheritance, encapsulation, and data abstraction and polymorphism, that enhance the code reusability and strengthen program stability and dependability.
2. **Multi – paradigm language** – Procedural, object – oriented and generic programming are among the several programming paradigms that C++ is compatible with.
3. **Portability** – With the combined features of an intermediate level of programming language, C++ codes require no alterations in order to be compiled and executed on numerous platforms and operating systems.
4. **Memory Management** – In addition to supporting inline functions, exception handling, pointers and references, C++ provides an efficient memory management and, enabling developers to optimize applications for high performance.
5. **Rich Library Support** – The C++ Standard Template Library (STL) provides a large framework that can be used to streamline and accelerate processes including access to database systems.

DISADVANTAGES OF C++

1. **Complexity** – Programmers who are unfamiliar with Object – Oriented Programming, C++ can be a challenging language to master.
2. **Lack of Built – in Support for Web Development** – Certain web development tasks such as AJAX and server – side scripting may require additional efforts to implement since C++ lacks the built – in support for such tasks.
3. **Absence of Integrated Multi – threading Support** – Developing parallel programs may become challenging due to a lack of built – in functionality for multi – threaded applications in C++.
4. **Security – related concerns** – Features such as pointers are vulnerable to null pointer dereferencing attacks, which may lead to program reliability issues.
5. **Potential Memory Leaks** – C++ programming language lacks the potential for garbage collection which causes the developers to manually manage memory allocation and deallocation, leading to an increase in the complexity of the code and memory leaks.

APPLICATIONS OF C++

1. **Web browsers and Operating Systems** – C++ is a fast and strongly-typed programming language which makes it an ideal choice for developing operating systems. A significant portion of Mac OS X, along with several softwares from Microsoft including Windows and IDE Visual Studio, is written in C++. Web browsers such as Mozilla Firefox are completely developed from C++ due to the need for rapid execution, low latency and high efficiency.
2. **Database Software** – C++ is used in developing database software, which requires efficient handling of data and performance.
3. **Banking Applications** – Since banking applications require concurrency and high performance, C++ is the default choice of programming language. Infosys Finacle is a popular banking application developed using C++.
4. **Embedded Systems** – C++ is used in various embedded systems, such as smartwatches and medical devices, where performance and efficiency are crucial.
5. **In-game Programming** – C++ is widely used in game development due to its speed, object-oriented programming capabilities, and ability to manipulate resources and override the complexities of 3D games efficiently.

DATA TYPES IN C++

All variables use data type during declaration to restrict the type of data to be stored.

C++ supports the following data types –

1. Primitive Data Types –

Also termed as primary / fundamental data types, these are built – in data types that can be directly used by the user to declare variables.

The operator `sizeof()` is used to find the number of bytes occupied by the data type in the computer memory.

Data Type	Keyword	Description	Memory (in bytes)	Example
Boolean	bool	It stores only 2 values – (a) true (returns 1) (b) false (returns 0).	1	bool val = true;
Character	char	It stores a single character. The character must be surrounded by single quotes. Alternatively, ASCII values may also be used to display certain characters.	1	char grade = 'A'; char a = 65;
Integer	int	It stores a whole number without decimals.	2 or 4	int num = 5;
Floating point	float	It stores floating point numbers (with decimals). Precision = 6 – 7 decimal digits.	4	float num = 9.75;
Double Floating Point	double	It stores floating point numbers (with decimals). Precision = 15 decimal digits.	8	double num = 10.345;

2. Derived Data Types – Data types derived from the primitive or built – in data types are referred to as Derived Data Types. These data types comprise of –

- (a) functions
- (b) arrays
- (c) pointers
- (d) references

3. User – defined Data Types – Also termed as Abstract data types, user – defined data types such as – class, structure, union and enumeration – are defined by the user itself.

HEADER FILE AND FUNCTIONS

C++ uses a convenient abstraction called streams, which are sequences of bytes that allow data to be transferred between devices and memory. They are used to perform input and output operations in sequential media such as the screen, the keyboard, or a file.

The **<iostream>** header file is a part of the C++ standard library and is used for input and output operations. It defines the standard input/output stream objects, such as cin, cout, cerr, and clog. Including this header file in a C++ program allows you to perform input and output operations using these stream objects.

Key features of the <iostream> header file include:

Object	Description	Operator used	Syntax	Output
cout	It is a standard output stream object, usually connected to the display screen. It is used to output values/print text.	Extraction (<<)	cout << "Hello World";	Hello World
cin	It is a standard input stream object, usually connected to the keyboard. It is used to accept the input from the standard input device.	Insertion (>>)	int a; cout << "Enter a number :"; cin >> a; cout << "a = " << a;	Enter a number : <u>10</u> a = 10
endl	It is a stream manipulator and a pre – defined object used to insert a newline character and flush the output stream, ensuring that any buffered output is displayed immediately.	Extraction (<<)	int a = 10; cout << "Num1: " << endl; cout << a;	Num1: 10

PROGRAMS:

Question No. 1 –

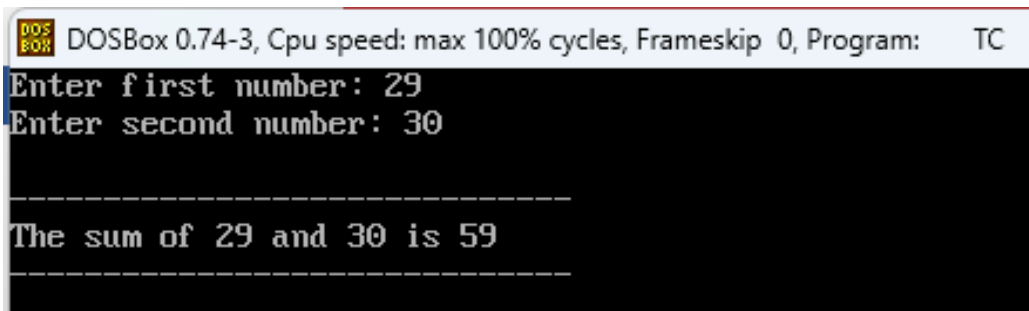
Write a program in C++ to print the sum of two numbers entered by the user.

CODE:

```
#include<iostream.h>
#include<conio.h>

void main()
{
    clrscr();
    int x, y, sum;
    cout << "Enter first number: ";
    cin >> x;
    cout << "Enter second number: ";
    cin >> y;
    sum = a + b;
    cout << "\n-----";
    cout << "\nThe sum of "<< x << " and "<< y << " is " << sum;
    cout << "\n-----";
    getch();
}
```

OUTPUT:



```
DOSBox 0.74-3, Cpu speed: max 100% cycles, Frameskip 0, Program: TC
Enter first number: 29
Enter second number: 30
-----
The sum of 29 and 30 is 59
-----
```

Figure 1: Printing the sum of two numbers entered by the user

Question No. 2 –

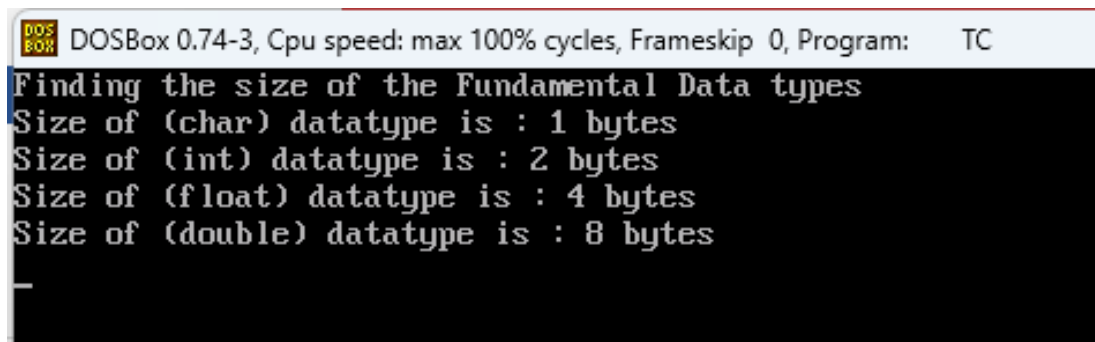
Write a program in C++ to find the size of the fundamental data types.

CODE:

```
#include<iostream.h>
#include<conio.h>

void main()
{
    clrscr();
    cout << "Finding the size of the Fundamental Data types" << endl;
    cout << "Size of (char) datatype is : "<< sizeof(char) << " bytes" << endl;
    cout << "Size of (int) datatype is : " << sizeof(int) << " bytes"<< endl;
    cout << "Size of (float) datatype is : " << sizeof(float) <<" bytes"<< endl;
    cout << "Size of (double) datatype is : " << sizeof(double) << " bytes"<< endl;
    getch();
}
```

OUTPUT:

A screenshot of a DOSBox window titled "DOSBox 0.74-3, Cpu speed: max 100% cycles, Frameskip 0, Program: TC". The terminal output shows the program's execution results: "Finding the size of the Fundamental Data types", "Size of (char) datatype is : 1 bytes", "Size of (int) datatype is : 2 bytes", "Size of (float) datatype is : 4 bytes", and "Size of (double) datatype is : 8 bytes". The text is displayed in a monospaced font on a black background.

```
DOSBox 0.74-3, Cpu speed: max 100% cycles, Frameskip 0, Program: TC
Finding the size of the Fundamental Data types
Size of (char) datatype is : 1 bytes
Size of (int) datatype is : 2 bytes
Size of (float) datatype is : 4 bytes
Size of (double) datatype is : 8 bytes
_
```

Figure 2: Finding the memory size of the fundamental data types

Question No. 3 –

Write a program in C++ to swap two numbers.

CODE:

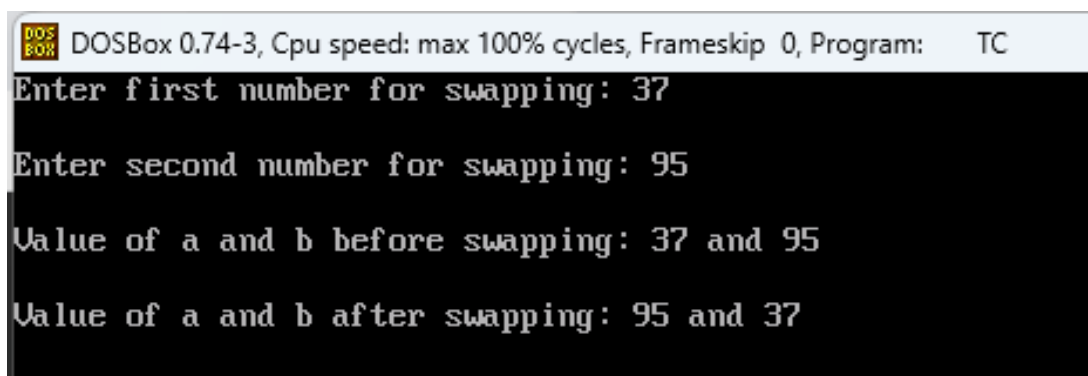
```
#include<iostream.h>
#include<conio.h>

void main()
{
    clrscr();
    int a, b, c;
    cout << "Enter first number for swapping: ";
    cin >> a;
    cout << "Enter second number for swapping: ";
    cin >> b;
    cout << "Value of a and b before swapping: "<<a<< " and " <<b<<endl;

    c = a;
    a = b;
    b = c;

    cout<<"Value of a and b after swapping: "<<a<< " and " <<b<<endl;
    getch();
}
```

OUTPUT:



```
DOSBox 0.74-3, Cpu speed: max 100% cycles, Frameskip 0, Program: TC
Enter first number for swapping: 37
Enter second number for swapping: 95
Value of a and b before swapping: 37 and 95
Value of a and b after swapping: 95 and 37
```

Figure 3: Swapping two numbers entered by the user

Question No. 4 –

Write a program in C++ to calculate the volume of a cube.

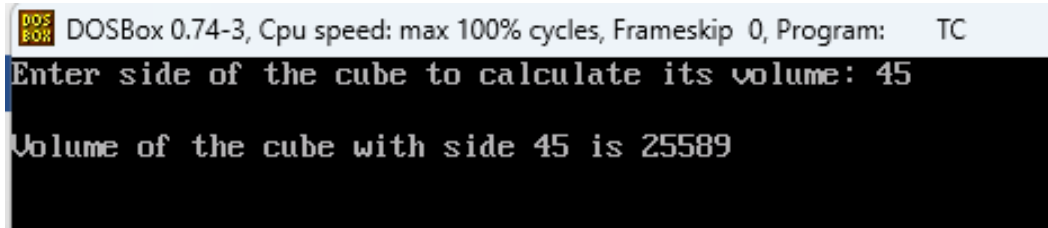
CODE:

```
#include<iostream.h>
#include<conio.h>

void main()
{
    clrscr();
    int side;
    cout << "Enter side of the cube to calculate its volume: ";
    cin >> side;
    cout << "\nVolume of the cube with side " << side << " is " <<
    (side*side*side) << endl;

    getch();
}
```

OUTPUT:



```
DOS
BOX DOSBox 0.74-3, Cpu speed: max 100% cycles, Frameskip 0, Program: TC
Enter side of the cube to calculate its volume: 45
Volume of the cube with side 45 is 25589
```

Figure 4: Calculating the Volume of a Cube

RESULTS:

Using the C++ programming language, the basic input and output operations were demonstrated using the <iostream.h> header file in order to display the sum of two user – entered numbers, display the memory size of the fundamental data types, swap two user – entered numbers and calculate the volume of a cube.

CONCLUSION:

Using the C++ programming language, the basic input and output operations were demonstrated using the <iostream.h> header file comprising of the ‘cin’ and ‘cout’ objects.

REFERENCES:

1. *Introduction to C Programming Language*. (2023, August 23). GeeksforGeeks. <https://www.geeksforgeeks.org/introduction-to-c-programming-language/>
 2. *Difference between cerr and clog*. (2023, November 2). GeeksforGeeks. <https://www.geeksforgeeks.org/difference-between-cerr-and-clog/>
 3. *A Guide to C++: Advantages and Disadvantages* | Pangea.ai. (2023, January 30). <https://pangea.ai/blog/languages/a-comprehensive-guide-to-c-advantages-and-disadvantages>
 4. S. (2023, August 4). *Top 7 Practical Applications of C++ and the Way to Build a Career*. Simplilearn.com. <https://www.simplilearn.com/c-plus-plus-programming-for-beginners-article>
 5. *Standard library header <iostream>* - cppreference.com. (n.d.). <https://en.cppreference.com/w/cpp/header/iostream>
 6. *Basic Input Output in C*. (2023, September 14). GeeksforGeeks. <https://www.geeksforgeeks.org/basic-input-output-c/>
-

Practical 3

C++ : Conditional Statements and Loops

AIM:

To demonstrate the use of conditional statements and loops in C++ programming language.

INTRODUCTION:

C++ is an object-oriented, multi-paradigm language that was developed as an extension of the C programming language and was designed for system and application programming. It supports procedural, functional, and generic programming styles.

In C++ programming language, conditional statements and loops are the fundamental concepts essential for controlling the flow of a C++ program and for implementing repetitive tasks. They are crucial for comprehending and developing efficient and effective C++ programs.

CONDITIONAL STATEMENTS

Conditional statements, also referred to as decision control structures, are utilized for decision – making purposes in C++ programs. These conditional statements are employed to assess and evaluate one or more conditions / criteria and determine whether or not to execute a series of statements or a specific block of code. Such decision – making statements in programming languages decide the direction of the flow of program execution.

Types of Conditional Statements

1. if Statement –

The **if** conditional statement is used to decide whether or not a specific statement or block of statements will be executed if a particular condition is true.

Syntax – if (condition)

```
{  
    // code to be executed if the condition returns true  
}
```

Diagram –

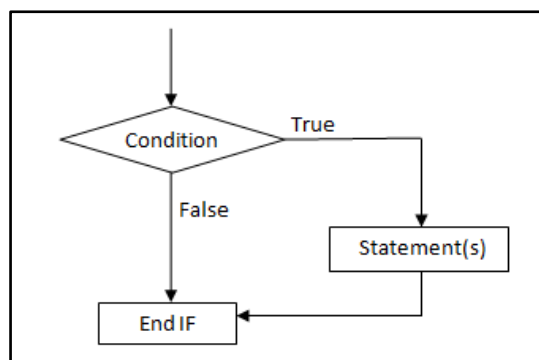


Figure 1: Working of **if** conditional statement

2. if – else Statement –

The **if - else conditional statement** is a two – way branching statement. It consists of two blocks of statements each enclosed inside **if block** and **else block** respectively. If the condition inside **if statement** is true, statements inside **if block** are executed, otherwise statements inside **else block** are executed.

Syntax – if (condition)

```
{  
    // code to be executed if the condition returns true  
}  
else  
{  
    // code to be executed if the condition returns false  
}
```

Diagram –

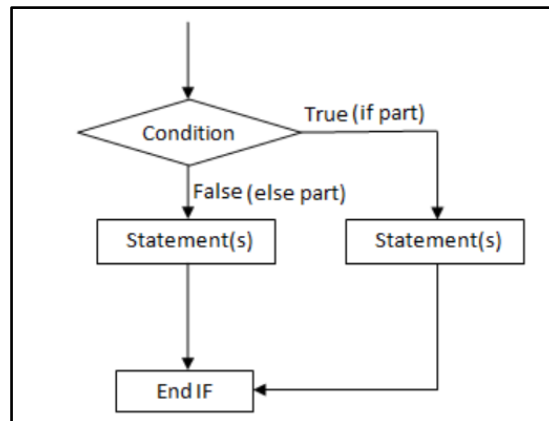


Figure 2: Working of **if – else** conditional statement

3. if – else – if Ladder –

The **if – else – if Ladder** is used when more than one condition is to be checked. A block of statement is enclosed inside **if**, **else if** and **else** part. Conditions are checked in each **if** and **else if** part. If the condition is true, the statements inside that block are executed. If none of the conditions are true, the statements inside **else** block are executed. Such a statement must have only one **if block** but can have as many **else if block** as required.

Syntax – if (condition1)

```
{  
    // code to be executed if condition1 returns true  
}  
else if (condition2)  
{  
    // code to be executed if condition1 returns false  
    // and condition2 returns true  
}
```

```

else if (conditionN)
{
    // code to be executed if previous conditions return false
    // and conditionN returns true
}
else
{
    // code to be executed if all the previous conditions return false
}

```

Diagram –

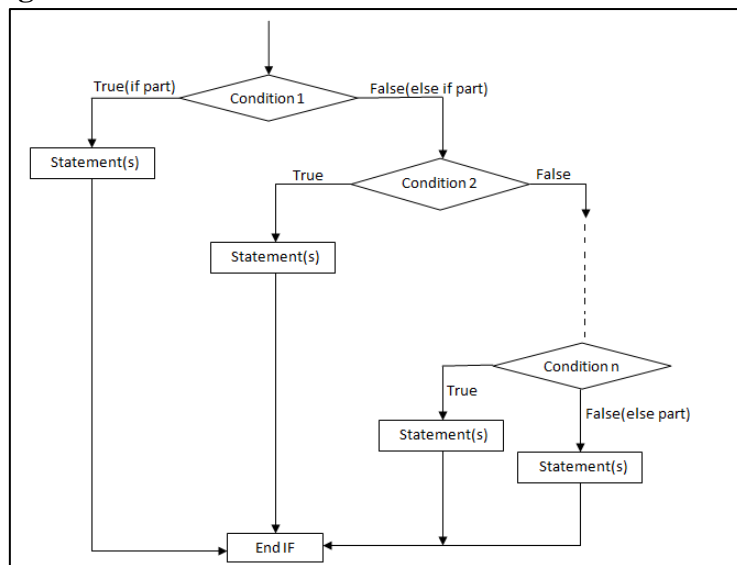


Figure 3: Working of if – else – if Ladder

4. Nested if Statement –

When an **if statement** is kept inside another **if statement**, it is called nested if statement. **Nested if statements** are used if there is a sub condition to be tested.

Syntax – if (condition1)

```

{
    // Nested if statement to be executed if condition1 returns true
    if (sub_condition1)
    {
        // code to be executed if sub_condition1 returns true
    }
}
else if (condition2)
{
    // code to be executed if condition1 returns false
    // and condition2 returns true
}

```

```

// Nested if statement to be executed if condition2 returns true
if (sub_condition2)
{
    // code to be executed if sub_condition2 returns true
}
}

```

Diagram –

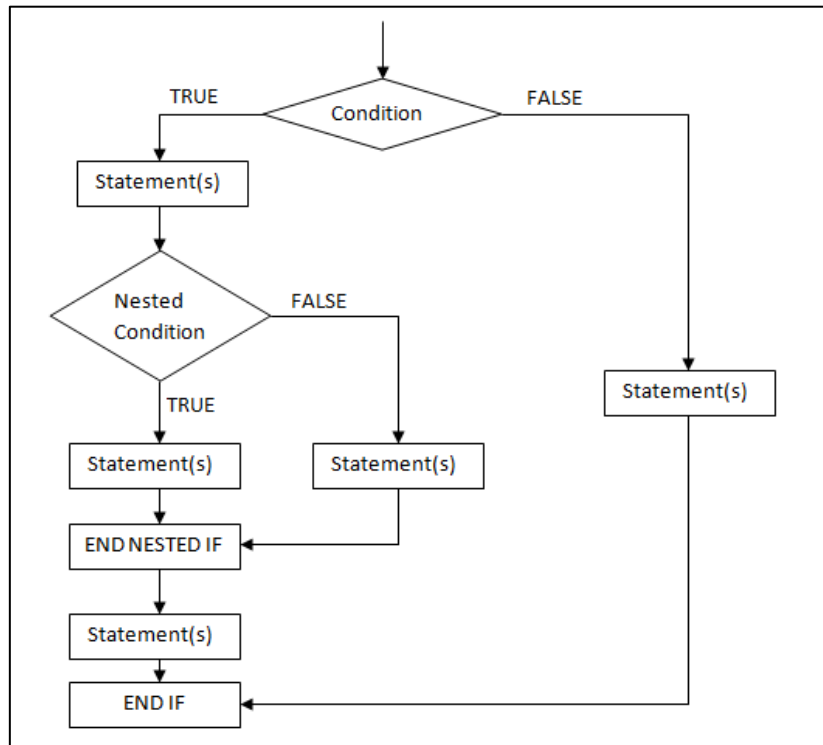


Figure 4: Working of Nested if conditional statement

LOOPS

Loops enable the repeated execution of a sequence of instructions until a predetermined condition is met. Following are the parts of a loop –

1. **Initialization Expression** – initializes the loop variables in the beginning of the loop.
2. **Test Expression** – decides whether the loop will be executed (if the test expression is true) or not (if the test expression is false).
3. **Update Expressions** – update (increment / decrement) the values of the loop variables after every iteration of the loop.
4. **Body of the loop** – contains a set of statements to be executed repeatedly.

Types of Loops

1. Entry – Controlled Loops –

In this type of loop, the test expression is checked before entering the body of the loop.

(a) for loop –

A **for loop** is a repetition control structure that allows us to write a loop that is executed a specific / pre – determined number of times. The loop enables us to perform ‘n’ number of steps sequentially in one line.

Syntax – for (initialization_expression ; test_expression ; update_expression)
{
 // code to be executed
}

Diagram –

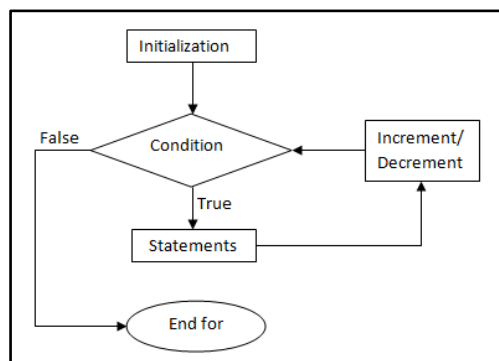


Figure 5: Working of for loop

(b) while loop –

A **while loop** repeats a statement or group of statements until a given condition is true. It tests the condition before executing the loop body. The loop consists of the test expression whereas the initialization expression and update expressions are addressed elsewhere.

Syntax – initialization_expression;
while(test_expression)
{
 // code to be executed
 update_expression;
}

Diagram –

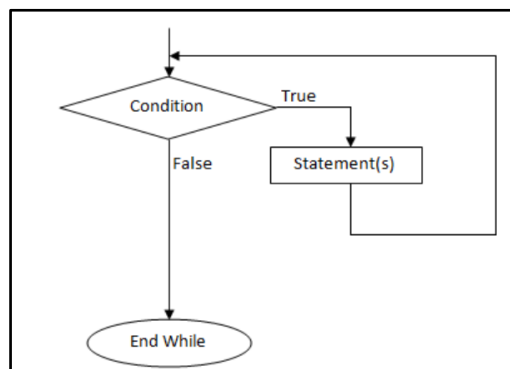


Figure 6: Working of while loop

2. Exit – Controlled Loops –

In this type of loop, the test condition is checked at the end of the loop body. Therefore, the loop body will execute a set of statements at least once, irrespective of whether the test expression is true or false.

do – while loop is a type of exit – controlled loop where the test expression is tested at the end of the loop body and the loop is terminated on the basis of the test conditions after executing the loop body at least once.

```
Syntax – initialization_expression;  
do  
{  
    // code to be executed  
    update_expression;  
} while(test_expression);
```

Diagram –

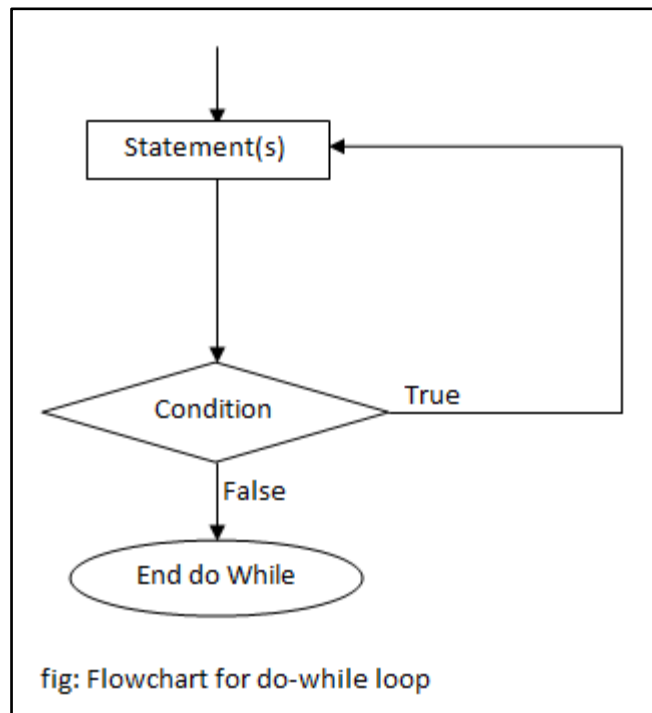


Figure 7: Working of do – while loop

PROGRAMS:

Question No. 1 –

Write a program in C++ that takes a number as input and prints its multiplication table upto 10.

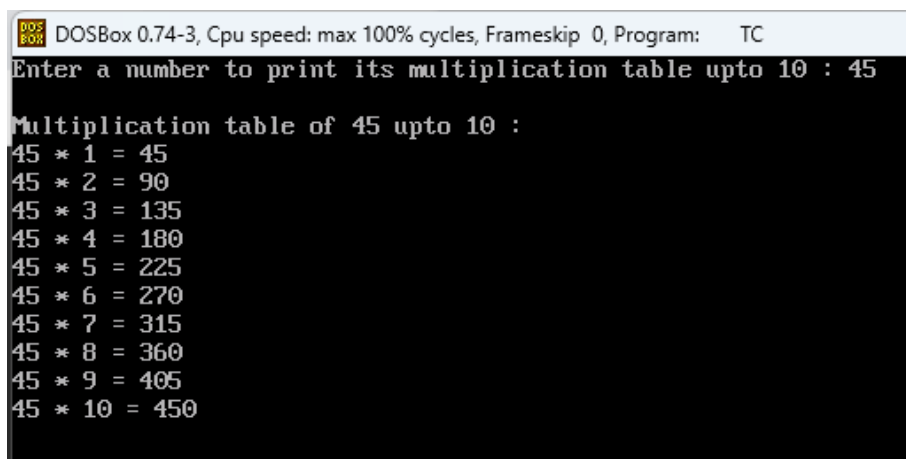
CODE:

```
#include<iostream.h>
#include<conio.h>

void main()
{
    clrscr();
    int i,a;
    cout << "Enter a number to print its multiplication table upto 10 : ";
    cin >> a;
    cout << "Multiplication table of " << a << " upto 10 : " << endl;

    for(i=1;i<=10;i++)
    {
        cout<<a << " * " << i << " = " << (a*i) <<endl;
    }
    getch();
}
```

OUTPUT:



```
DOSBox 0.74-3, Cpu speed: max 100% cycles, Frameskip 0, Program: TC
Enter a number to print its multiplication table upto 10 : 45
Multiplication table of 45 upto 10 :
45 * 1 = 45
45 * 2 = 90
45 * 3 = 135
45 * 4 = 180
45 * 5 = 225
45 * 6 = 270
45 * 7 = 315
45 * 8 = 360
45 * 9 = 405
45 * 10 = 450
```

Figure 8: Displays the multiplication table of the number entered by the user upto 10

Question No. 2 –

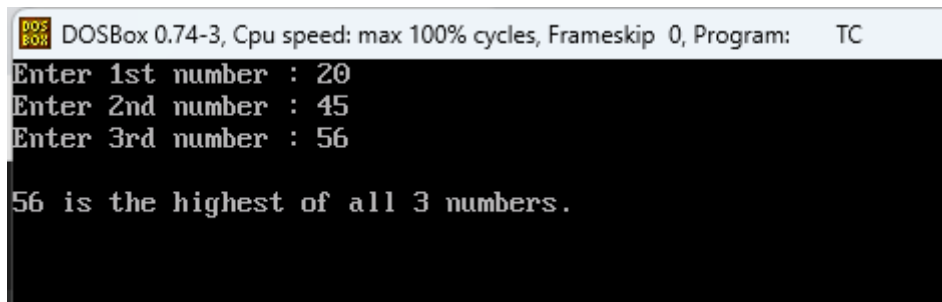
Write a C++ program which prints the highest numbers from entered 3 numbers.

CODE:

```
#include<iostream.h>
#include<conio.h>

void main()
{
    clrscr();
    int a, b, c;
    cout<<"Enter 1st number : ";
    cin >> a;
    cout<<"Enter 2nd number : ";
    cin >> b;
    cout<<"Enter 3rd number : ";
    cin >> c;
    if(a>b && a>c)
    {
        cout<<"\n" << a << " is the highest of all 3 numbers." <<endl;
    }
    else if(b>a && b>c)
    {
        cout<<"\n" << b << " is the highest of all 3 numbers." <<endl;
    }
    else if(c>a && c>b)
    {
        cout<<"\n" << c << " is the highest of all 3 numbers." <<endl;
    }
    getch();
}
```

OUTPUT:



```
DOSBox 0.74-3, Cpu speed: max 100% cycles, Frameskip 0, Program: TC
Enter 1st number : 20
Enter 2nd number : 45
Enter 3rd number : 56
56 is the highest of all 3 numbers.
```

Figure 9: Displays the highest numbers from the 3 numbers entered by the user

Question No. 3 –

Write a C++ program to compute the sum of even numbers.

CODE:

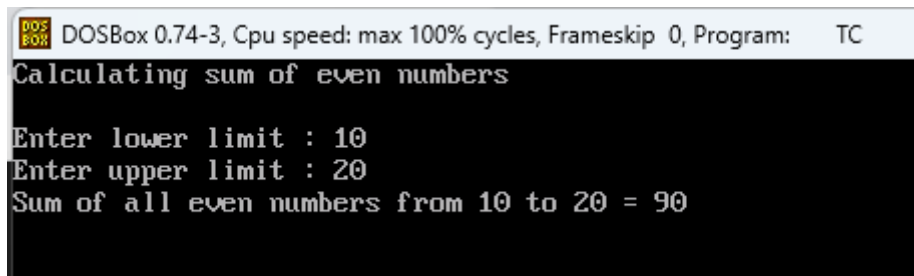
```
#include<iostream.h>
#include<conio.h>

void main()
{
    clrscr();
    int i, upperlimit, lowerlimit, sum = 0;
    cout<<"Calculating sum of even numbers"<<endl;
    cout<<"\nEnter lower limit : ";
    cin>>lowerlimit;
    cout<<"Enter upper limit : ";
    cin>>upperlimit;

    for(i=lowerlimit ; i<=upperlimit ; i+=2)
    {
        sum += i;
    }

    cout<<"Sum of all even numbers from " <<lowerlimit<<" to " << upperlimit
    << " = " << sum << endl;
    getch();
}
```

OUTPUT:



```
DOSBox 0.74-3, Cpu speed: max 100% cycles, Frameskip 0, Program: TC
Calculating sum of even numbers
Enter lower limit : 10
Enter upper limit : 20
Sum of all even numbers from 10 to 20 = 90
```

Figure 10: Calculates and displays the sum of even numbers between a particular range entered by the user

Question No. 4 –

Write a C++ program to accept the three sides of a triangle and display equilateral, isosceles and scalene triangle.

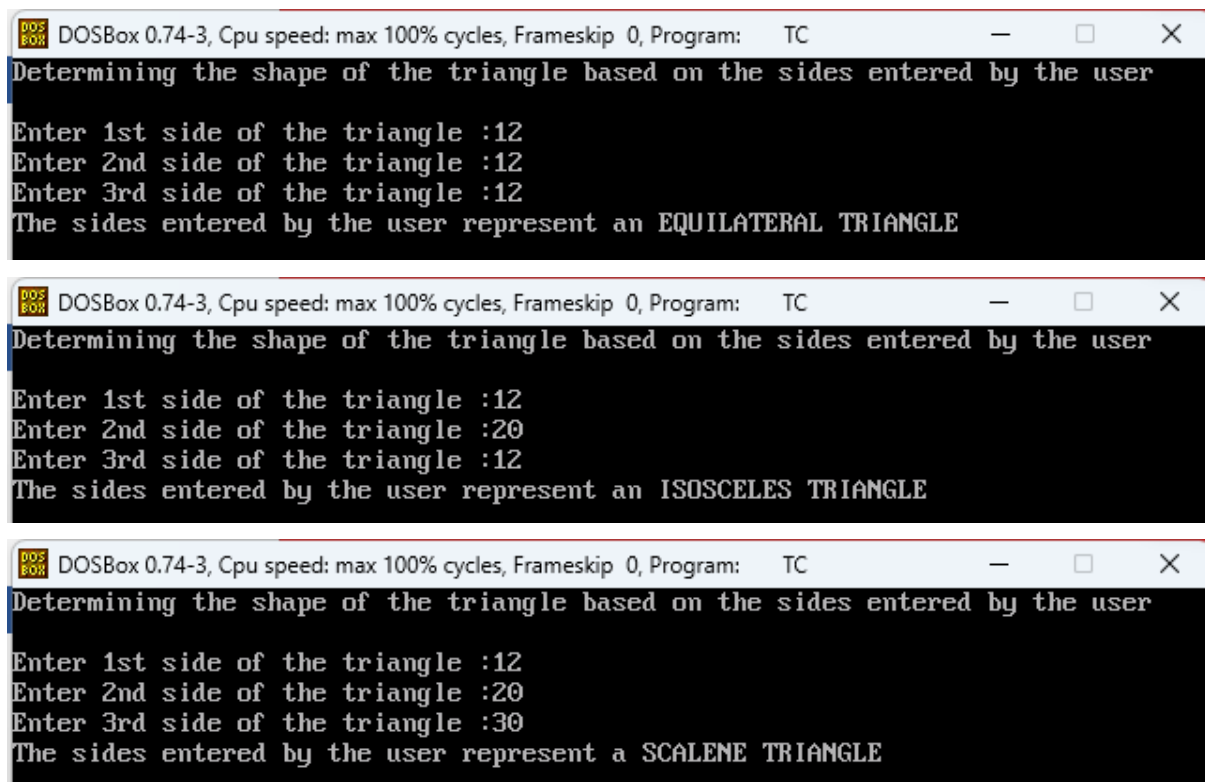
CODE:

```
#include<iostream.h>
#include<conio.h>

void main()
{
    clrscr();
    int side1, side2, side3;
    cout<<"Determining the shape of the triangle based on the sides entered by the user" << endl;
    cout<<"Enter 1st side of the triangle :";
    cin>>side1;
    cout<<"Enter 2nd side of the triangle :";
    cin>>side2;
    cout<<"Enter 3rd side of the triangle :";
    cin>>side3;

    if (side1 == side2 && side2 == side3 && side3 == side1)
    {
        cout<<"The sides entered by the user represent an EQUILATERAL TRIANGLE";
    }
    else if (side1 == side2 || side1 == side3 || side2 == side3)
    {
        cout<<"The sides entered by the user represent an ISOSCELES TRIANGLE";
    }
    else
    {
        cout<<"The sides entered by the user represent a SCALENE TRIANGLE";
    }
    getch();
}
```

OUTPUT:



The figure consists of three vertically stacked screenshots of a DOSBox window. Each window has a title bar that reads "DOSBox 0.74-3, Cpu speed: max 100% cycles, Frameskip 0, Program: TC". The main content area of each window displays the following text:

```
Determining the shape of the triangle based on the sides entered by the user  
Enter 1st side of the triangle :12  
Enter 2nd side of the triangle :12  
Enter 3rd side of the triangle :12  
The sides entered by the user represent an EQUILATERAL TRIANGLE
```

```
Determining the shape of the triangle based on the sides entered by the user  
Enter 1st side of the triangle :12  
Enter 2nd side of the triangle :20  
Enter 3rd side of the triangle :12  
The sides entered by the user represent an ISOSCELES TRIANGLE
```

```
Determining the shape of the triangle based on the sides entered by the user  
Enter 1st side of the triangle :12  
Enter 2nd side of the triangle :20  
Enter 3rd side of the triangle :30  
The sides entered by the user represent a SCALENE TRIANGLE
```

Figure 11: Identifying and displaying the type of triangle based on the value of the sides entered by the user

RESULTS:

Using the C++ programming language, the use of conditional statements and loops was demonstrated in order to print the multiplication table of a user – entered number upto 10, print the highest of 3 numbers, compute the sum of even numbers and identify the type of triangle based on the value of the sides of the triangle entered by the user.

CONCLUSION:

Using the C++ programming language, the use of conditional statements and loops was demonstrated which aid in directing the flow of the C++ programs.

REFERENCES:

1. Team, U. (2021, February 24). *C++ Loops: What You Need to Know*. Udacity. <https://www.udacity.com/blog/2021/02/c-loops-what-you-need-to-know.html>
2. P. (2021, January 15). *for loop in C++ Programming - Programtopia*. Programtopia. <https://www.programtopia.net/cplusplus/docs/for-loop>
3. *Decision Making in C C if if..else Nested if if else if.* (2023, November 15). GeeksforGeeks. <https://www.geeksforgeeks.org/decision-making-c-cpp/>
4. *C Loops.* (2023, July 6). GeeksforGeeks. <https://www.geeksforgeeks.org/cpp-loops/>
5. *C++ If . . . Else.* (n.d.). https://www.w3schools.com/cpp/cpp_conditions.asp

Practical 4

C++ : Arrays (One – Dimensional and Two – Dimensional Arrays)

AIM:

To demonstrate the use of arrays in C++ programming language.

INTRODUCTION:

C++ is an object-oriented, multi-paradigm language that was developed as an extension of the C programming language and was designed for system and application programming. It supports procedural, functional, and generic programming styles.

In C++ programming language, an **array** is a data structure that is used to store multiple values of similar data types in a contiguous memory location. A number of arithmetic and dynamic programming methods rely on arrays as their efficient means of implementing data structures like lists and vectors. Essentially, arrays are used to store intermediate solutions of a more complex problem as well as to perform sorting and searching operations. Matrix operations, cryptography, data mining and signal processing are few of the major applications where arrays can be employed efficiently.

Following are the properties of an array in the C++ programming language –

1. Indexing of an array commences from 0, hence storing the first element at the 0th index, the second at the 1st, and so on. Hence, Elements of an array can be individually referenced and modified using their indices, which range from 0 to (size_of_the_array -1). For instance, arr[3] retrieves the fourth element of the array 'arr'.
2. The size of an array remains constant throughout the program post declaration of the array. The operator **sizeof** can be used to determine the number of elements in an array.
3. An array can have multiple dimensions.

Property	Syntax	Example
Declaration of an Array		
Declaration of an Array	data_type array_name [size_of_the_array];	int arr[5];
Initialization of an Array		
(a) With Values	data_type array_name [size_of_the_array] = {element1, element2, element3....elementN}; // Number of elements are equal to the size of the array	int arr[5] = {1, 2, 3, 4, 5};

(b) With Values and Without Size	data_type array_name[] = {element1, element2, element3....elementN };	int arr[] = {1, 2, 3, 4, 5};
(c) After Declaration using loops	for (i = 0 ; i <= size_of_array ; i++) { arr[i] = element_value; }	int n = 10; int value = 1; for (int i = 0 ; i <= n ; i++) { arr[i] = value; value++; }
(d) Partial Initialization	data_type array_name [size_of_the_array] = {element1, element2, element3....elementN}; // Number of elements are less than the size of the array	int partial_arr[5] = {1, 2};

TYPES OF ARRAYS

1. One – dimensional Arrays –

A one – dimensional array is a group of elements having the same datatype which are stored in a linear arrangement under a single variable name. It requires only one indices specification in order to access a particular element of the array.

Representation: They represent multiple data items as a list.

Total Memory (in bytes) = Size of (base type) * Size of the array

Syntax for Declaration – data_type array_name [size_of_the_array]

Example for Declaration and Initialization: int arr[3] = {1, 2, 3};

2. Two – dimensional Arrays –

Also termed as a ‘matrix’, two – dimensional arrays are the simplest form of multi – dimensional arrays in which each array is itself an element. It requires two index specifications (row number and column number) in order to access a particular element of the array.

Representation: They represent multiple data items in a tabular format consisting of rows and columns.

Total Memory (in bytes) =

Size of (base type) * (Number of rows) * (Number of columns)

Syntax for Declaration –

data_type array_name [number_of_rows][number_of_columns]

Example for Declaration and Initialization: int arr[2][5] = { {1, 2, 3, 4, 5},
{6, 7, 8, 9, 10}};

PROGRAMS:

Question No. 1 –

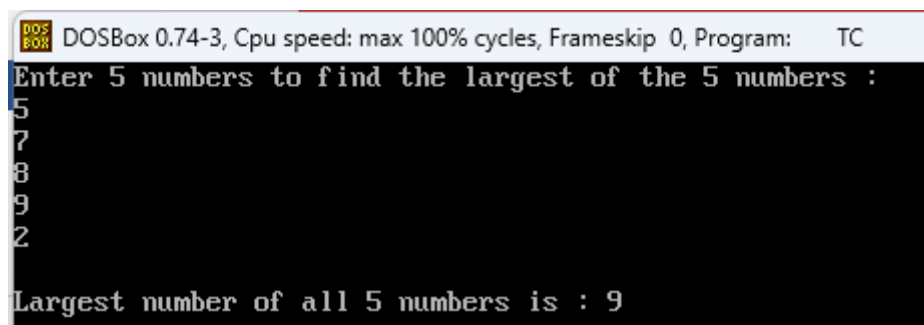
Write a C++ program to find the largest element from a given array of integers.

Array: {5, 7, 8, 9, 2}.

CODE:

```
#include<iostream.h>
#include<conio.h>
void main()
{
    clrscr();
    int m[5];
    int i, j, a, largest;
    cout << "Enter 5 numbers to find the largest of the 5 numbers : "<< endl;
    for(i = 0 ; i <= 4 ; i++)
    {
        cin >> m[i];
    }
    largest = m[0];
    for(j = 0 ; j <=4 ; j++)
    {
        a = m[j];
        if(a > largest)
        {
            largest = a;
        }
    }
    cout << endl;
    cout << "Largest number of all 5 numbers is : " << largest << endl;
    getch();
}
```

OUTPUT:



```
DOSBox 0.74-3, Cpu speed: max 100% cycles, Frameskip 0, Program: TC
Enter 5 numbers to find the largest of the 5 numbers :
5
7
8
9
2
Largest number of all 5 numbers is : 9
```

Figure 1: Finding the largest element from a given array of integers

Question No. 2 –

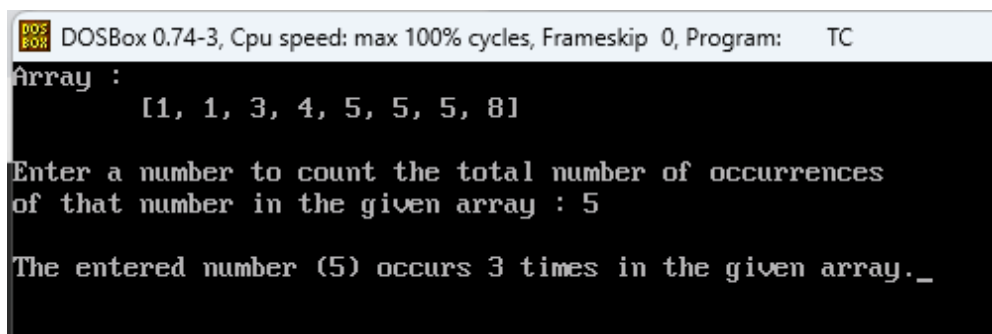
Write a C++ program to count the number of occurrences of a given number in an array of integers. Array: {1, 1, 3, 4, 5, 5, 5, 8}

User entered number: 5 ; Count displayed: 3

CODE:

```
#include<iostream.h>
#include<conio.h>
void main()
{
    clrscr();
    int num[8] = {1,1,3,4,5,5,5,8};
    int count = 0, n, i;
    cout << "Array : " << endl;
    cout << "\t[";
    for(i=0 ; i<=6 ; i++)
    {
        cout << num[i] << " , ";
    }
    cout << num[7];
    cout << "]" << endl << endl;
    cout << "Enter a number to count the total number of occurrences" << endl <<
    "of that number in the given array : ";
    cin >> n;
    for(i=0 ; i<=7; i++)
    {
        if(num[i]==n)
        {
            count++;
        }
    }
    cout << endl;
    cout << "The entered number (" << n << ") occurs " << count << " times in
    the given array.";
    getch();
}
```

OUTPUT:



```
DOSBox 0.74-3, Cpu speed: max 100% cycles, Frameskip 0, Program: TC
Array :
    [1, 1, 3, 4, 5, 5, 5, 8]
Enter a number to count the total number of occurrences
of that number in the given array : 5
The entered number (5) occurs 3 times in the given array._
```

Figure 2: Counting the number of occurrences of a given number in an array of integers

Question No. 3 –

Write a C++ program to display all the even numbers from an entered array.

Array: {2, 5, 9, 12, 4}

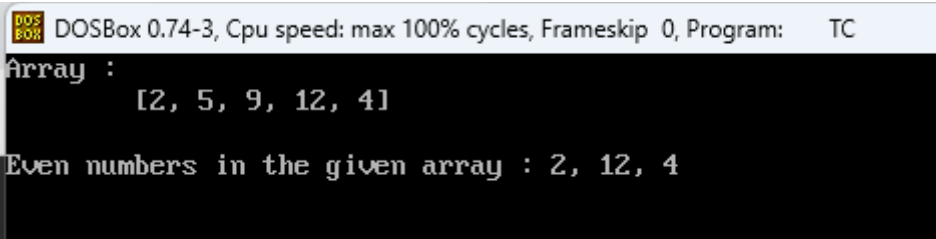
Even numbers in the array: 2, 12, 4

CODE:

```
#include<iostream.h>
#include<conio.h>

void main()
{
    clrscr();
    int num[5] = {2, 5, 9, 12, 4};
    int i;
    cout << "Array : " << endl << "\t[";
    for(i=0 ; i<=3 ; i++)
    {
        cout << num[i] << ", ";
    }
    cout << num[4] << "]" << endl << endl;
    cout << "Even numbers in the given array : ";
    for(i=0 ; i<=3 ; i++)
    {
        if(num[i] % 2 == 0)
        {
            cout << num[i] << ", ";
        }
    }
    if(num[4] % 2 == 0)
    {
        cout << num[4] << endl;
    }
    getch();
}
```

OUTPUT:



```
DOSBox 0.74-3, Cpu speed: max 100% cycles, Frameskip 0, Program: TC
Array :
    [2, 5, 9, 12, 4]
Even numbers in the given array : 2, 12, 4
```

Figure 3: Displaying all even numbers from an entered array

Question No. 4 –

Write a C++ program to accept 3x3 matrix and display the transpose of a given matrix.

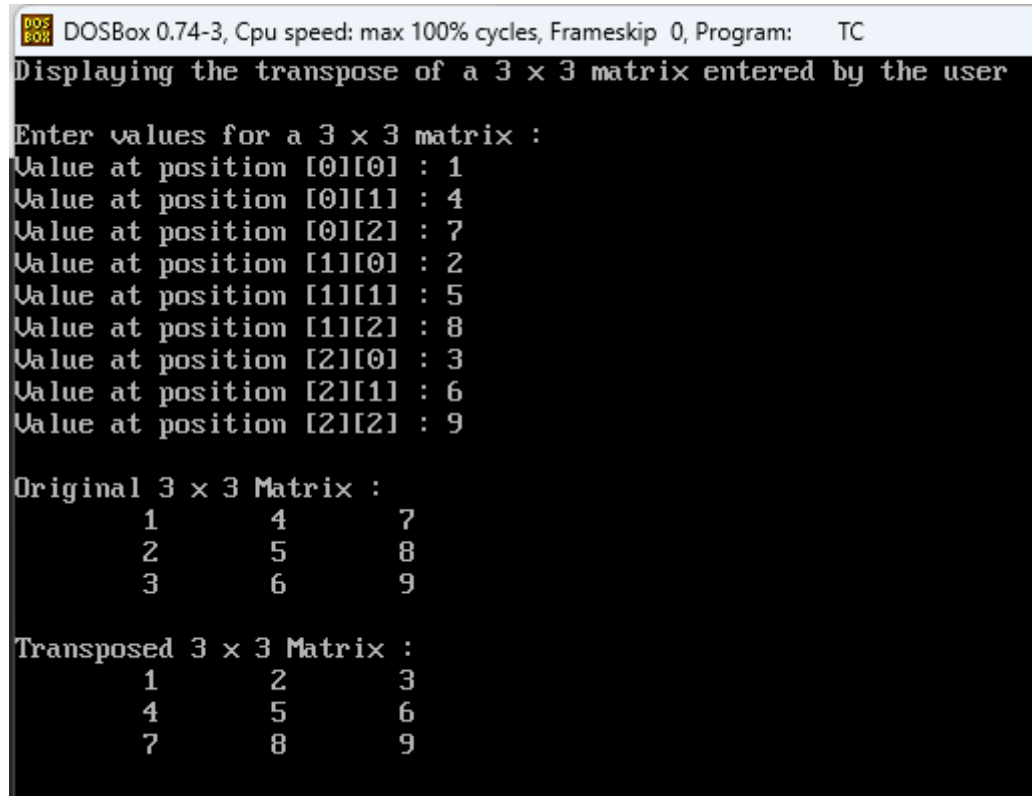
CODE:

```
#include<iostream.h>
#include<conio.h>
void main()
{
    clrscr();
    int matrix[3][3];
    int i, j;
    cout << "Displaying the transpose of a 3 x 3 matrix entered by the user" <<
    endl << endl;
    cout << "Enter values for a 3 x 3 matrix : " << endl;
    for(i=0 ; i<=2 ; i++)
    {
        for(j=0 ; j<=2 ; j++)
        {
            cout << "Value at position [" << i << "][" << j << " ] : ";
            cin >> matrix[i][j];
        }
    }
    cout << endl;
    cout << "Original 3 x 3 Matrix : " << endl;
    cout << "\t";
    for(i=0 ; i<=2 ; i++)
    {
        for(j=0 ; j<=2; j++)
        {
            cout << matrix[i][j] << "\t";
        }
        cout << endl << "\t";
    }
    cout << endl;

    cout << "Transposed 3 x 3 Matrix : " << endl;
    cout << "\t";
    for(i=0 ; i<=2 ; i++)
    {
        for(j=0 ; j<=2; j++)
        {
            cout << matrix[j][i] << "\t";
        }
        cout << endl << "\t";
    }
```

```
}  
cout << endl;  
getch();  
}
```

OUTPUT:



The screenshot shows a DOSBox window titled "DOSBox 0.74-3, Cpu speed: max 100% cycles, Frameskip 0, Program: TC". The program prompts the user to enter values for a 3x3 matrix. It then displays the original matrix and its transpose.

```
DOSBox 0.74-3, Cpu speed: max 100% cycles, Frameskip 0, Program: TC  
Displaying the transpose of a 3 x 3 matrix entered by the user  
  
Enter values for a 3 x 3 matrix :  
Value at position [0][0] : 1  
Value at position [0][1] : 4  
Value at position [0][2] : 7  
Value at position [1][0] : 2  
Value at position [1][1] : 5  
Value at position [1][2] : 8  
Value at position [2][0] : 3  
Value at position [2][1] : 6  
Value at position [2][2] : 9  
  
Original 3 x 3 Matrix :  
    1    4    7  
    2    5    8  
    3    6    9  
  
Transposed 3 x 3 Matrix :  
    1    2    3  
    4    5    6  
    7    8    9
```

Figure 4: Displaying the transpose of a user – entered 3 x 3 matrix

Question No. 5 –

Write a C++ program to accept two 3x3 matrices and display the sum of two matrices.

CODE:

```
#include<iostream.h>
#include<conio.h>
void main()
{
    clrscr();
    int matrix1[3][3];
    int matrix2[3][3];
    int sum[3][3];
    int i, j;
    cout << "Addition of two 3 x 3 matrices entered by the user" << endl << endl;
    cout << "Enter values for Matrix 1 : " << endl;
    for(i=0 ; i<=2 ; i++)
    {
        for(j=0 ; j<=2 ; j++)
        {
            cout << "Value at position [" << i << "]" << j << " : ";
            cin >> matrix1[i][j];
        }
    }
    cout << endl;

    cout << "Enter values for Matrix 2 : " << endl;
    for(i=0 ; i<=2 ; i++)
    {
        for(j=0 ; j<=2 ; j++)
        {
            cout << "Value at position [" << i << "]" << j << " : ";
            cin >> matrix2[i][j];
        }
    }
    cout << endl;
    cout << "Matrix 1 : " << endl;
    cout << "\t";

    for(i=0 ; i<=2 ; i++)
    {
        for(j=0 ; j<=2; j++)
        {
            cout << matrix1[i][j] << "\t";
        }
        cout << endl << "\t";
    }
    cout << endl;
```

```

cout << "Matrix 2 : " << endl;
cout << "\t";

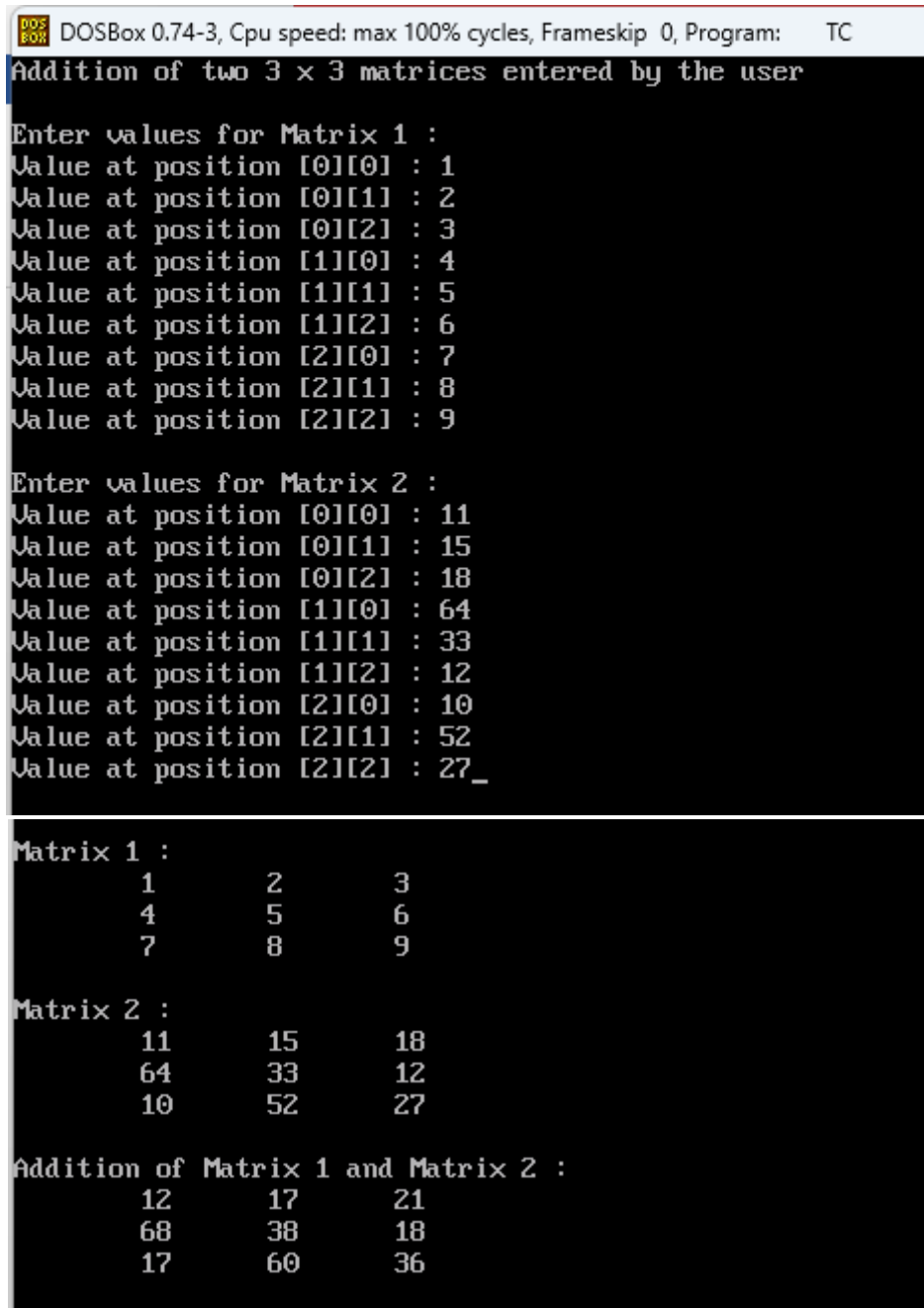
for(i=0 ; i<=2 ; i++)
{
    for(j=0 ; j<=2; j++)
    {
        cout << matrix2[i][j] << "\t";
    }
    cout << endl << "\t";
}
cout << endl;

for(i=0 ; i<=2 ; i++)
{
    for(j=0 ; j<=2 ; j++)
    {
        sum[i][j] = matrix1[i][j] + matrix2[i][j];
    }
}

cout << "Addition of Matrix 1 and Matrix 2 : " << endl;
cout << "\t";
for(i=0 ; i<=2 ; i++)
{
    for(j=0 ; j<=2; j++)
    {
        cout << sum[i][j] << "\t";
    }
    cout << endl << "\t";
}
cout << endl;
getch();
}

```

OUTPUT:



```
DOSBox 0.74-3, Cpu speed: max 100% cycles, Frameskip 0, Program: TC
Addition of two 3 x 3 matrices entered by the user

Enter values for Matrix 1 :
Value at position [0][0] : 1
Value at position [0][1] : 2
Value at position [0][2] : 3
Value at position [1][0] : 4
Value at position [1][1] : 5
Value at position [1][2] : 6
Value at position [2][0] : 7
Value at position [2][1] : 8
Value at position [2][2] : 9

Enter values for Matrix 2 :
Value at position [0][0] : 11
Value at position [0][1] : 15
Value at position [0][2] : 18
Value at position [1][0] : 64
Value at position [1][1] : 33
Value at position [1][2] : 12
Value at position [2][0] : 10
Value at position [2][1] : 52
Value at position [2][2] : 27_

Matrix 1 :
    1    2    3
    4    5    6
    7    8    9

Matrix 2 :
    11   15   18
    64   33   12
    10   52   27

Addition of Matrix 1 and Matrix 2 :
    12   17   21
    68   38   18
    17   60   36
```

Figure 5: Displaying the Addition of two user – entered 3 x 3 matrices

RESULTS:

Using the C++ programming language, both one – dimensional and two – dimensional arrays were created to store data elements entered by the user and further manipulated and modified to organize data in order to display the largest element, count the total number of occurrences of an element and all the even numbers from a given array of integers, to display the transpose of a user – entered 3x 3 matrix and to display the addition of two user – entered 3 x 3 matrices.

CONCLUSION:

Using the C++ programming language, the use of arrays was demonstrated by storing and modifying data elements through one – dimensional and two – dimensional arrays.

REFERENCES:

1. Kirch-Prinz, U., & Prinz, P. (2002, January 1). *A Complete Guide to Programming in C++*. Jones & Bartlett Learning.
 2. *C++ Arrays (With Examples)*. (n.d.). <https://www.programiz.com/cpp-programming/arrays>
 3. *Difference Between one dimensional and two dimensional array*. (2023, November 2). GeeksforGeeks. <https://www.geeksforgeeks.org/difference-between-one-dimensional-and-two-dimensional-array/>
 4. C++ Arrays. (n.d.). https://www.w3schools.com/cpp/cpp_arrays.asp
-

Practical 5

C++ : Functions and Structures

AIM:

To demonstrate the use of functions and structures in C++ programming language.

INTRODUCTION:

C++ is an object-oriented, multi-paradigm language that was developed as an extension of the C programming language and was designed for system and application programming. It supports procedural, functional, and generic programming styles.

In C++ programming language, a **function** is a segment or a block of code that performs a specific task. It is a fundamental building block of C++ programs, allowing programmers to modularize their code and make it more reusable, maintainable, and understandable. Functions enable programmers to divide complex programs into smaller, more manageable units. This modular approach promotes code reuse, as functions can be easily incorporated into different parts of the program or even reused across multiple projects. The reusability of functions significantly enhances code development efficiency.

By providing a structured approach to code organization and encapsulating a specific task within functions, programmers can avoid rewriting the same code repeatedly, which not only saves time but also reduces the risk of errors and improves code maintainability by making it easier to identify and isolate issues. Functions play a crucial role in enhancing code readability. By clearly defining the purpose and scope of each code segment, functions enhance code readability. Hence, functions are essential tools for C++ programmers, enabling them to create structured, reusable, and maintainable code. By effectively utilizing functions, programmers can write well – organized, efficient, and easy – to – understand C++ programs.

Syntax for Declaration and Definition of a Function –

```
return_type function_name(parameters) // function header
{
    // function body
    // code to be executed
}
```

TYPES OF FUNCTIONS

1. Built – in functions –

Also termed as Standard Library functions, built – in functions are a component of a previously defined compiler package. These functions can be accessed by the compiler from their specific header files and can be implemented in the program directly without defining them.

Built – in functions can be used for several purposes, including performing arithmetic operations and manipulating string objects.

Example – `sqrt()`, `strcat()`.

2. User – defined functions –

Also termed as tailor – made functions, user – defined functions are referred to custom – designed blocks of code created by the programmer to perform specific tasks. These user – defined functions are necessary in order to enhance the reusability of the code and reduce the complexity of a large program.

Following are the types of user – defined functions –

(a) Functions with No Return Value and No Parameter –

A function that does not take any input parameters / arguments and does not return any value. The return type of the function is ‘void’.

Syntax	Example
<pre>void function_name() { //code to be executed } void main() { // calling the function function_name(); getch(); }</pre>	<pre>void sum() { int a = 5, b = 10, c; c = a + b; cout << "Sum =" << c; } void main() { sum(); getch(); }</pre>

(b) Functions with No Return Value and With Parameter –

A function that takes one or more input parameters / arguments and does not return any value.

The return type of the function is 'void'.

These functions can be called in the main() function and parameters can be passed either by value or by reference.

Parameters passed during function call are termed as Actual Parameters.

Parameters received by the function are termed as Formal Parameters.

Syntax	Example
<pre>void function_name(data_type parameter) { //code to be executed } void main() { // calling the function function_name(parameter); getch(); }</pre>	<pre>void sum(int a, int b) { int c; c = a + b; cout << "Sum =" << c; } void main() { sum(5, 10); getch(); }</pre>

(c) Functions With Return Value and No Parameter –

A function that does not take any input parameters / arguments but returns a value using the keyword **return** after executing certain operations and performing a specific task.

The return type of the function is the same as the data type of the value to be returned.

Syntax	Example
<pre>return_type function_name() { //code to be executed // return statement return value; } void main() { // calling the function function_name(); getch(); }</pre>	<pre>int sum() { int a = 5, b = 10, c; c = a + b; return c; } void main() { int s = sum(); cout << "Sum =" << s; getch(); }</pre>

(d) Functions With Return Value and With Parameter –

A function that takes one or more input parameters / arguments and returns a value using the keyword **return** after executing certain operations and performing a specific task.

The return type of the function is the same as the data type of the value to be returned.

The functions can be called in the main() function and parameters can be passed either by value or by reference.

Syntax	Example
<pre>return_type function_name(data_type parameter) { //code to be executed // return statement return value; } void main() { //calling the function function_name(parameter); getch(); }</pre>	<pre>int sum(int a, int b) { int c; c = a + b; return c; } void main() { int s = sum(5, 10); cout << "Sum =" << s; getch(); }</pre>

PROGRAMS:

Question No. 1 –

Write a C++ program to create a function named as ‘area’ and find the area of a triangle using no return and no parameter.

CODE:

```
#include<iostream.h>
#include<conio.h>

void area()
{
    int b, h;
    cout << "Calculating Area of Triangle" << endl;
    cout << "Enter value of base : ";
    cin >> b;
    cout << "Enter value of height : ";
    cin >> h;
    float area = 0.5 * b * h;
    cout << "\nArea of the triangle = " << area;
}

void main()
{
    clrscr();
    area();
    getch();
}
```

OUTPUT:

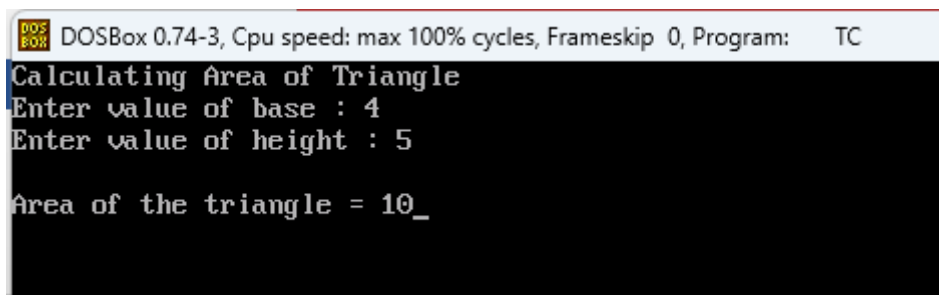


Figure 1: Finding the area of a triangle using a User – defined function with No Return value and No Parameter

Question No. 2 –

Write a C++ program to create a function for adding two numbers using with return and with parameter.

CODE:

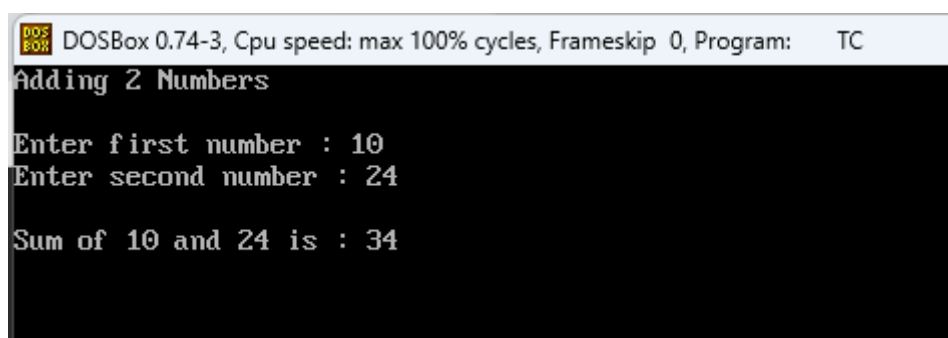
```
#include<iostream.h>
#include<conio.h>

int add(int x, int y)
{
    int sum = x + y;
    return sum;
}

void main()
{
    clrscr();
    int x, y;
    cout << "Adding 2 Numbers" << endl << endl;
    cout << "Enter first number : ";
    cin >> x;
    cout << "Enter second number : ";
    cin >> y;
    cout << "\nSum of " << x << " and " << y << " is : " << add(x,y);

    getch();
}
```

OUTPUT:



```
DOSBox 0.74-3, Cpu speed: max 100% cycles, Frameskip 0, Program: TC
Adding 2 Numbers
Enter first number : 10
Enter second number : 24
Sum of 10 and 24 is : 34
```

Figure 2: Displaying the sum of two numbers using a User – defined function With Return value and With Parameter

RESULTS:

Using the C++ programming language, the use of functions and structures was demonstrated by developing programs that consists of a no return value and no parameter user – defined function to find the area of a triangle in addition to a with return value and with parameter user – defined function to calculate and display the sum of two numbers.

CONCLUSION:

Using the C++ programming language, the use of functions and structures was demonstrated using user – defined functions.

REFERENCES:

1. *C++ Functions*. (n.d.). https://www.w3schools.com/cpp/cpp_functions.asp
 2. *Functions in C*. (2023, September 4). GeeksforGeeks. <https://www.geeksforgeeks.org/functions-in-cpp/>
 3. *C++ Functions with No Arguments and No return value*. (n.d.). <https://www.w3schools.in/cplusplus/examples/functions-with-no-arguments-no-return-value>
-

Practical 6

C++ : String Manipulation

AIM:

To demonstrate the use of string manipulation in C++ programming language.

INTRODUCTION:

C++ is an object-oriented, multi-paradigm language that was developed as an extension of the C programming language and was designed for system and application programming. It supports procedural, functional, and generic programming styles.

In C++ programming language, **strings** are fundamental data structures used to represent and manipulate sequences of characters. They are employed in a wide range of programming tasks, from simple text processing to complex data analysis and manipulation. C++ offers two primary methods for handling strings: **C – style character arrays** and the **std::string** class.

1. **C-style character arrays** represent strings as arrays of characters terminated by a null character (`\0`). They are the type of strings that C++ inherited from C language.
2. The **std::string** class, introduced in the C++ Standard Library and defined inside `<string.h>` header file, provides a powerful and versatile tool for managing strings. It offers a rich set of member functions for manipulating strings, including concatenation, substring extraction, character access, and formatting. This provides many advantages over conventional C-style strings such as dynamic size, member functions, etc.

Syntax for Declaring and Defining a String –

1. **Using the keyword 'string'** – `string variable_name = "string_to_be_stored";`
2. **Storing as an array of characters –**
`char variable_name[size] = {'char1', 'char2'....'charN', '\0'};`
`char variable_name[size] = "string_to_be_stored";`

Example for Declaring and Defining a String –

```
string s = "HelloWorld"; // using the string keyword
```

```
// stored as an array of characters
```

```
char s[ ] = {'H', 'e', 'l', 'l', 'o', 'W', 'o', 'r', 'l', 'd', '\0'};  
char s1[10] = "HelloWorld";
```

STRING MANIPULATION

String manipulation in C++ involves various operations such as concatenation, finding the length, searching for substrings, extracting substrings, and replacing parts of a string. String manipulation in C++ offers several advantages, including ease of use, compatibility, memory efficiency, performance overhead, and the ability to handle various data types.

Function	Description	Return type	Syntax
strcmp()	Used to compare two null – terminated strings lexicographically by comparing the ASCII value of each character. (a) If two strings are equal, it returns 0. (b) If first string is greater than the second string, it returns a positive integer. (c) If first string is less than the second string, it returns a negative integer.	Integer	strcmp(string1, string2);
strcpy()	Used to copy one string to another.	String	strcpy(string1, string2);
strlen()	Used to find the length of a string, including all the characters and spaces as well.	Integer	strlen(string);
strcat()	Used to append a copy of one string to the end of another string.	String	strcat(string1, string2);

PROGRAMS:

Question No. 1 –

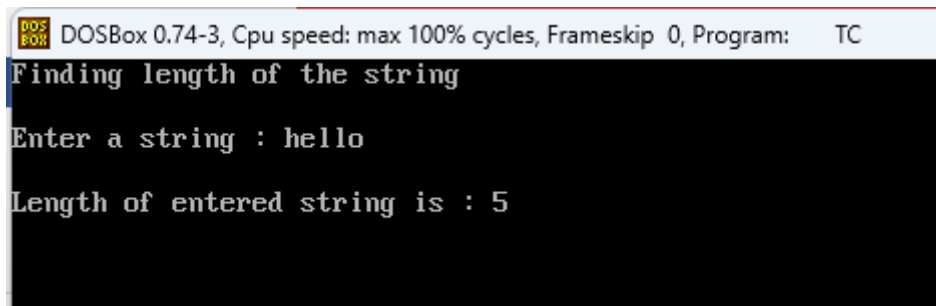
Write a C++ program to accept a string and find length of the string.

CODE:

```
#include<iostream.h>
#include<conio.h>
#include<string.h>

void main()
{
    clrscr();
    char string1[10];
    cout << "Finding length of the string" << endl;
    cout << "\nEnter a string : ";
    cin >> string1;
    cout << "\nLength of entered string is : " << strlen(string1);
    getch();
}
```

OUTPUT:



```
DOSBox 0.74-3, Cpu speed: max 100% cycles, Frameskip 0, Program: TC
Finding length of the string
Enter a string : hello
Length of entered string is : 5
```

Figure 1: Finding the length of the user – entered string

Question No. 2 –

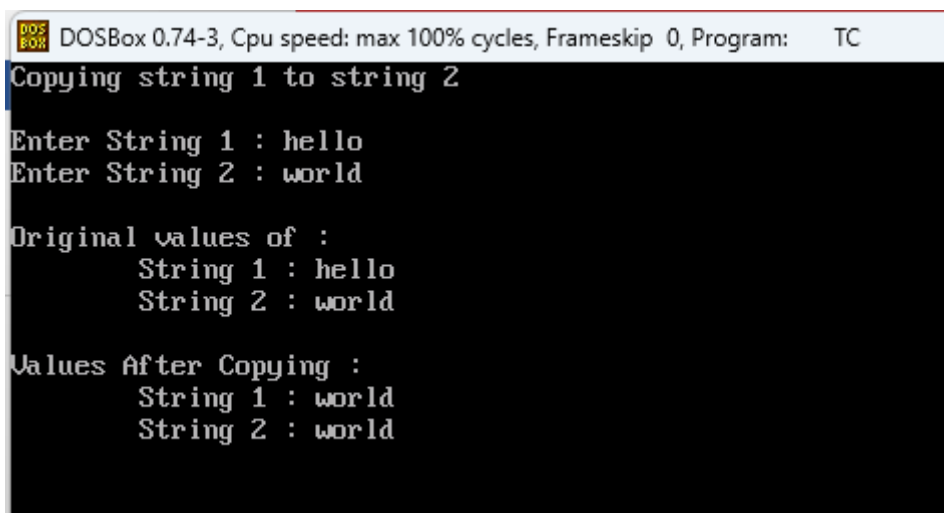
Write a C++ program to accept two strings and copy a string into another.

CODE:

```
#include<iostream.h>
#include<conio.h>
#include<string.h>

void main()
{
    clrscr();
    char string1[10];
    char string2[10];
    cout << "Copying string 1 to string 2" << endl;
    cout << "\nEnter String 1 : ";
    cin >> string1;
    cout << "Enter String 2 : ";
    cin >> string2;
    cout << "\nOriginal values of : " << endl << "\tString 1 : " << string1 << endl
    << "\tString 2 : " << string2 << endl;
    strcpy(string1, string2);
    cout << "\nValues After Copying : " << endl << "\tString 1 : " << string1 <<
    endl << "\tString 2 : " << string2 << endl;
    getch();
}
```

OUTPUT:



```
DOSBox 0.74-3, Cpu speed: max 100% cycles, Frameskip 0, Program: TC
Copying string 1 to string 2
Enter String 1 : hello
Enter String 2 : world

Original values of :
String 1 : hello
String 2 : world

Values After Copying :
String 1 : world
String 2 : world
```

Figure 2: Copying one user – entered string into another user – entered string

Question No. 3 –

Write a C++ program to accept two strings and combine the two strings.

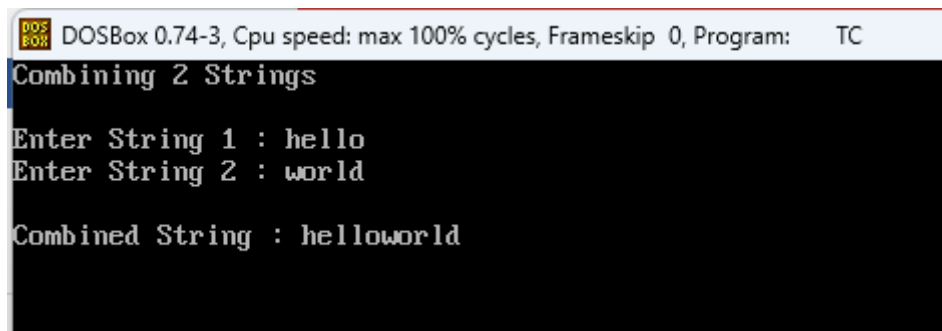
CODE:

```
#include<iostream.h>
#include<conio.h>
#include<string.h>

void main()
{
    clrscr();
    char string1[10];
    char string2[10];
    cout << "Combining 2 Strings" << endl;
    cout << "\nEnter String 1 : ";
    cin >> string1;
    cout << "Enter String 2 : ";
    cin >> string2;
    cout << "\nCombined String : " << strcat(string1, string2);

    getch();
}
```

OUTPUT:



```
DOSBox 0.74-3, Cpu speed: max 100% cycles, Frameskip 0, Program: TC
Combining 2 Strings
Enter String 1 : hello
Enter String 2 : world
Combined String : helloworld
```

Figure 3: Combining two user – entered strings

Question No. 4 –

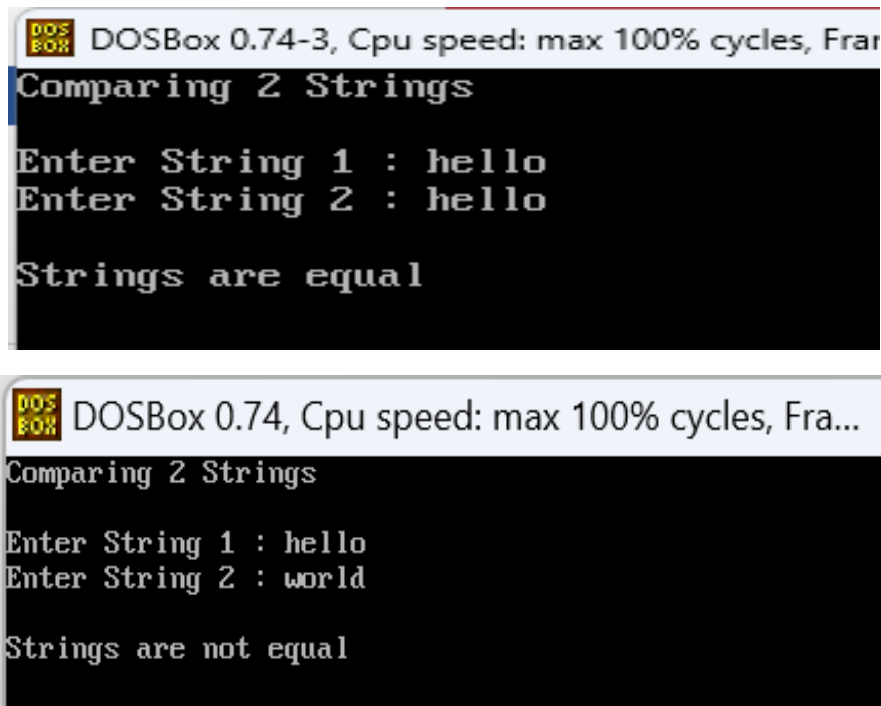
Write a C++ program to accept two strings and compare whether the two user – entered strings are equal or not.

CODE:

```
#include<iostream.h>
#include<conio.h>
#include<string.h>

void main()
{
    clrscr();
    char string1[10];
    char string2[10];
    cout << "Comparing 2 Strings" << endl;
    cout << "\nEnter String 1 : ";
    cin >> string1;
    cout << "Enter String 2 : ";
    cin >> string2;
    cout << endl;
    if(strcmp(string1, string2) == 0)
    {
        cout << "Strings are equal" << endl;
    }
    else
    {
        cout << "Strings are not equal" << endl;
    }
    getch();
}
```

OUTPUT:



The image contains two screenshots of a DOSBox terminal window. The top screenshot shows the text 'DOSBox 0.74-3, Cpu speed: max 100% cycles, Fra...' at the top, followed by 'Comparing 2 Strings'. Below this, it prompts 'Enter String 1 : hello' and 'Enter String 2 : hello', and finally displays 'Strings are equal'. The bottom screenshot shows 'DOSBox 0.74, Cpu speed: max 100% cycles, Fra...' at the top, followed by 'Comparing 2 Strings'. Below this, it prompts 'Enter String 1 : hello' and 'Enter String 2 : world', and finally displays 'Strings are not equal'.

```
DOSBox 0.74-3, Cpu speed: max 100% cycles, Fra...
Comparing 2 Strings
Enter String 1 : hello
Enter String 2 : hello
Strings are equal

DOSBox 0.74, Cpu speed: max 100% cycles, Fra...
Comparing 2 Strings
Enter String 1 : hello
Enter String 2 : world
Strings are not equal
```

Figure 4: Comparing whether two user – entered strings are equal or not

RESULTS:

Using the C++ programming language, the use of string manipulation was demonstrated using both the C – style character arrays and string class derived from the <string.h> header file to manipulate user – entered string objects in order to find their length, copy one string to another string, combine two strings and compare the two strings lexicographically to determine if the two strings are equal or not.

CONCLUSION:

Using the C++ programming language, the use of string manipulation was demonstrated using the C – style character arrays and string class retrieved from the <string.h> header file.

REFERENCES:

1. K. (2020, August 31). *C++ String Manipulation - C++ Tutorials*. C++ Tutorials. <https://www.kindsonthegenius.com/cplusplus/c-string-manipulation/>
2. *Strings in C*. (2023, October 9). GeeksforGeeks. <https://www.geeksforgeeks.org/strings-in-cpp/>
3. *std string class in C*. (2023, February 17). GeeksforGeeks. <https://www.geeksforgeeks.org/stdstring-class-in-c/>

Practical 7

C++ : Class, Objects and Encapsulation

AIM:

To explore and demonstrate the concept of classes, objects and encapsulation in C++ programming language.

INTRODUCTION:

C++ is an object-oriented, multi-paradigm language that was developed as an extension of the C programming language and was designed for system and application programming. It supports procedural, functional, and generic programming styles.

CLASS AND OBJECTS

A **class** in C++ programming language is the building block that leads to Object – Oriented programming. It is a user – defined data type, which holds its own data members and member functions, which can be accessed and used by creating an object (instance) of that class.

Data members are the data variables that represent the characteristics or properties of an object. They are essentially variables that store the unique values of each object.

Member functions are the functions used to manipulate these variables together. They are essentially functions that encapsulate the operations specific to that class. These data members and member functions define the properties and behaviour of the objects in a class.

Objects represent individual instances of a class, embodying the characteristics and behaviours defined within that class. An object is an instantiation of the class blueprint. Objects possess their own unique values for the attributes defined in the class.

C++ class and objects provide a structured way to organize data and behaviour, promoting code reuse, data encapsulation, and modular design. Classes and objects offer several advantages in C++ programming –

1. **Data Encapsulation** – Classes encapsulate data and behaviour, preventing unauthorized access and promoting data integrity.
2. **Code Reusability** – Classes promote code reuse by allowing us to define common characteristics and behaviours once and then create multiple objects with those characteristics and behaviours.
3. **Modular Design** – Classes contribute to modular design by breaking down complex programs into smaller, manageable units.

Access Specifiers / Access Modifiers that specify the access rights for the class members include –

1. **public** – Members are accessible from outside the class by other classes and functions. They can be accessed using the direct member access operator (.) with the object of that class.
2. **protected** – Members cannot be accessed outside the class. Although, class members declared as protected can be accessed by any subclass (derived class) of that class.

- 3. private** – Members cannot be accessed / viewed from outside of the class. They can be accessed only by the member functions inside the class.

By default, access to members of a C++ class is private.

Syntax for Declaration and Definition of Class and Objects –

```
class class_name
{
    Access_specifier: // private, public or protected
    data_members;

    member_functions( )
    {
        // code to be executed
    }
};
```

ENCAPSULATION

One of the core fundamental concepts of object-oriented programming (OOP) in C++ is encapsulation, which is the grouping of methods and data members into a single class. By limiting access to private data members from outside the class, encapsulation aids in data hiding. It provides the following advantages –

- 1. Data Protection** – Encapsulation restricts direct access to class members, preventing unauthorized modification or misuse of data.
- 2. Improved Code Readability and Maintainability** – By organizing data and functions together, encapsulation makes code more understandable and easier to manage.
- 3. Modular Design** – Encapsulation promotes modularity by dividing code into distinct, self-contained units, enhancing code reusability and reducing complexity.
- 4. Security Enhancement** – Encapsulation helps protect sensitive data by restricting access to authorized methods, preventing unauthorized modifications or data breaches.

Encapsulation plays a crucial role in OOP by promoting data integrity, improving code organization, and enhancing security. It is an essential concept for writing well-structured, maintainable, and secure C++ code.

PROGRAMS:

Question No. 1 –

Write a program to create class Area to calculate area of circle using private variables and public functions.

CODE:

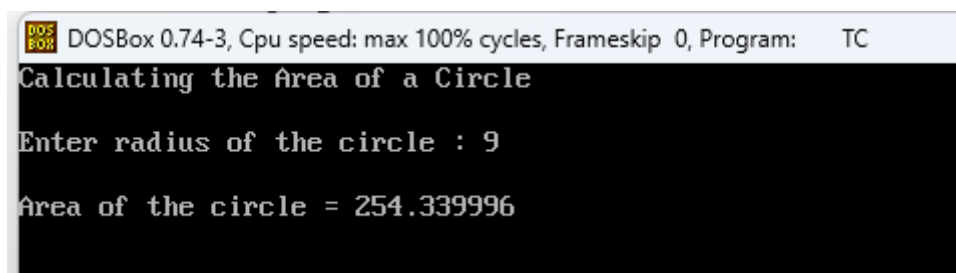
```
#include<iostream.h>
#include<conio.h>

class Area
{
    private:
        float area;
        int r;

    public:
        void areaOfCircle()
        {
            cout << "Calculating the Area of a Circle" << endl << endl;
            cout << "Enter radius of the circle : ";
            cin >> r;
            area = 3.14 * r * r;
            cout << "\nArea of the circle = " << area << endl;
        }
};

void main()
{
    clrscr();
    Area a;
    a.areaOfCircle();
    getch();
}
```

OUTPUT:



```
DOSBox 0.74-3, Cpu speed: max 100% cycles, Frameskip 0, Program: TC
Calculating the Area of a Circle
Enter radius of the circle : 9
Area of the circle = 254.339996
```

Figure 1: Calculating and displaying the area of a circle using private variables and public functions

RESULTS:

Using the C++ programming language, the concept of classes, objects and encapsulation was explored and demonstrated by creating a class to calculate the area of a circle using private variables and public functions.

CONCLUSION:

Using the C++ programming language, the concept of classes, objects and encapsulation was explored and demonstrated.

REFERENCES:

1. *C Classes and Objects*. (2023, April 17). GeeksforGeeks. <https://www.geeksforgeeks.org/c-classes-and-objects/>
 2. *C++ Access Specifiers*. (n.d.). https://www.w3schools.com/cpp/cpp_access_specifiers.asp
 3. Team, U. (2021, September 3). *C++ Encapsulation: An Overview*. Udacity. <https://www.udacity.com/blog/2021/09/cpp-encapsulation-an-overview.html>
 4. *Encapsulation in C*. (2023, September 4). GeeksforGeeks. <https://www.geeksforgeeks.org/encapsulation-in-cpp/>
-

Practical 8

C++ : Inheritance

AIM:

To explore and demonstrate the concept of inheritance in C++ programming language.

INTRODUCTION:

C++ is an object-oriented, multi-paradigm language that was developed as an extension of the C programming language and was designed for system and application programming. It supports procedural, functional, and generic programming styles.

In C++, **inheritance** is a fundamental concept in object-oriented programming (OOP) that allows new classes to be created from existing classes. The new class created is called “derived class” or “child class” and the existing class is known as the “base class” or “parent class”. The derived class is said to be inherited from the base class. The derived class inherits all the properties of the base class, without changing the properties of base class and may add new features to its own. These new features in the derived class will not affect the base class.

Inheritance promotes code reusability by allowing programmers to leverage the functionality of existing classes when creating new ones. This reduces the need to rewrite code from scratch, saving time and effort. Inheritance also promotes code organization by grouping related classes together based on their shared properties and behaviours. Inheritance enables the creation of hierarchical class structures that makes code more understandable, easier to manage and reflect real – world relationships between entities.

Base Class Visibility	Derived Class Visibility		
	Public Derivation	Private Derivation	Protected Derivation
private	Not inherited	Not inherited	Not inherited
protected	protected	private	protected
public	public	private	protected

Syntax –

```

class base_class_name
{
    // access specifiers
    // data members and member functions
};

class derived_class_name : access_specifier base_class_name
{
    // access specifiers
    // data members and member functions of the derived class
};

```

```
void main( )  
{  
    // create object of derived class to access data members and member functions  
    // of both the base class and derived class  
}
```

TYPES OF INHERITANCE

1. **Single Inheritance** – In single inheritance, a derived class inherits from only one base class. This ensures a clear parent – child relationship and avoiding potential conflicts.
2. **Multiple Inheritance** – In multiple inheritance, a derived class inherits from multiple base classes. This allows the derived class to combine the features of multiple base classes, creating more complex and specialized classes. However, multiple inheritance can introduce complexity and potential conflicts, making it less commonly used.
3. **Multilevel Inheritance** – Multilevel inheritance is a type of inheritance in C++ where a class inherits from another class, which in turn inherits from another class. This creates a hierarchical structure of classes, where each level inherits the properties and methods of the classes below it.
4. **Hierarchical Inheritance** – Hierarchical inheritance involves multiple derived classes inheriting from a common base class, forming a tree – like structure. This reflects a parent – child relationship between classes, with derived classes becoming increasingly specialized as they inherit from their ancestors.
5. **Hybrid Inheritance** – Hybrid inheritance combines multiple inheritance and hierarchical inheritance, creating complex relationships between classes. While it provides flexibility, it can also introduce significant complexity.

PROGRAMS:

Question No. 1 –

Write a program to perform single inheritance to calculate area of triangle.

CODE:

```
#include<iostream.h>
#include<conio.h>

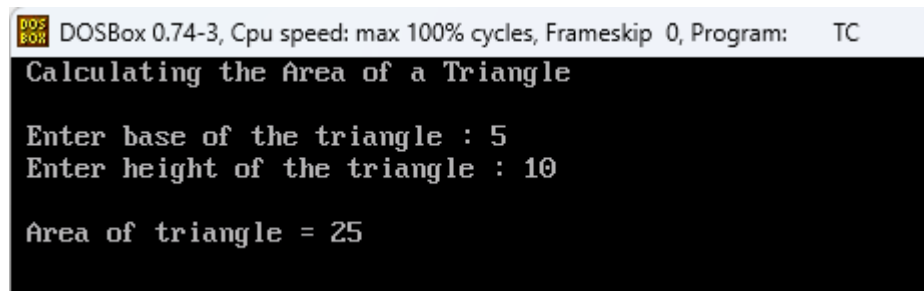
class Area
{
    protected:
        int b, h;

    public:
        void inputArea()
        {
            cout << " Calculating the Area of a Triangle" << endl << endl;
            cout << " Enter base of the triangle : ";
            cin >> b;
            cout << " Enter height of the triangle : ";
            cin >> h;
        }
};

class CalculateArea : public Area
{
    public:
        void calculate()
        {
            float area = 0.5 * b * h;
            cout << endl << " Area of triangle = " << area << endl;
        }
};

void main()
{
    clrscr();
    CalculateArea a;
    a.inputArea();
    a.calculate();
    getch();
}
```

OUTPUT:

A screenshot of a DOSBox window. The title bar reads "DOSBox 0.74-3, Cpu speed: max 100% cycles, Frameskip 0, Program: TC". The window content shows a C++ program's output in a monospaced font. It starts with "Calculating the Area of a Triangle", followed by prompts "Enter base of the triangle : 5" and "Enter height of the triangle : 10". The final output line is "Area of triangle = 25".

```
DOSBox 0.74-3, Cpu speed: max 100% cycles, Frameskip 0, Program: TC
Calculating the Area of a Triangle

Enter base of the triangle : 5
Enter height of the triangle : 10

Area of triangle = 25
```

Figure 1: Calculating and displaying the area of a triangle using single inheritance

Question No. 2 –

Write a program to perform multilevel inheritance using student details, course details, marks.

CODE:

```
#include<iostream.h>
#include<conio.h>
#include<string.h>

class StudentDetails
{
    public:
        char s_name[30];
        char s_address[30];
        int s_rollno;

    public:
        void inputStudentDetails()
        {
            cout << " STUDENT DETAILS, COURSE DEATILS AND
            MARKS" << endl << endl;
            cout << " ENTER STUDENT DETAILS" << endl << endl;
            cout << " Enter Name of the Student : ";
            cin >> s_name;
            cout << " Enter Roll No. of the Student : ";
            cin >> s_rollno;
            cout << " Enter Address of the Student : ";
            cin >> s_address;
        }
};
```

```

class CourseDetails : public StudentDetails
{
    protected:
        char c_name[30];
        char c_code[30];

    public:
        void inputCourseDetails()
        {
            cout << "\n\n ENTER COURSE DETAILS" << endl << endl;
            cout << " Enter Name of the Course : ";
            cin >> c_name;
            cout << " Enter Course Code : ";
            cin >> c_code;
        }
};

class Marks : public CourseDetails
{
    protected:
        int it_marks, cellbio_marks, bioinfo_marks, total;
        float percentage;

    public:
        void inputandCalculateMarks()
        {
            cout << "\n\n ENTER MARKS" << endl << endl;
            cout << " Enter Marks obtained in IT : ";
            cin >> it_marks;
            cout << " Enter Marks obtained in Cell Biology : ";
            cin >> cellbio_marks;
            cout << " Enter Marks obtained in Bioinformatics : ";
            cin >> bioinfo_marks;
            total = it_marks + cellbio_marks + bioinfo_marks;
            percentage = (total/3);
        }

        void displayAllDetails()
        {
            cout << "\n\n*****" << endl;
            cout << "\tFINAL RESULT " << endl;
            cout << "*****" << endl << endl;
            cout << " Student Name : " << s_name << endl;
            cout << " Student Roll No. : " << s_rollno << endl;
            cout << " Student Address : " << s_address << endl;
        }
};

```



```

        cout << " Course Name : " << c_name << endl;
        cout << " Course Code : " << c_code << endl << endl;
        cout << " Marks Obtained in : " << endl;
        cout << " \tIT : " << it_marks << endl;
        cout << " \tCell Biology : " << cellbio_marks << endl;
        cout << " \tBioinformatics : " << bioinfo_marks << endl <<
        endl;
        cout << " Total Marks Obtained : " << total << " / 300" <<
        endl;
        cout << " Percentage Obtained : " << percentage << " %" <<
        endl;
    }
};

void main()
{
    clrscr();
    Marks obj;
    obj.inputStudentDetails();
    obj.inputCourseDetails();
    obj.inputandCalculateMarks();
    obj.displayAllDetails();
    getch();
}

```

OUTPUT:

```

DOSBox 0.74-3, Cpu speed: max 100% cycles, Frameskip 0, Program: TC
STUDENT DETAILS, COURSE DEATILS AND MARKS

ENTER STUDENT DETAILS

Enter Name of the Student : Elizabeth
Enter Roll No. of the Student : 115
Enter Address of the Student : Mumbai

ENTER COURSE DETAILS

Enter Name of the Course : MScBioinformatics
Enter Course Code : GNKPBIOINFO101

ENTER MARKS

Enter Marks obtained in IT : 100
Enter Marks obtained in Cell Biology : 95
Enter Marks obtained in Bioinformatics : 98_

```

Figure 2: Accepting student details, course details and marks from the user

```
DOSBox 0.74-3, Cpu speed: max 100% cycles, Frameskip 0, Program: TC
ENTER MARKS

Enter Marks obtained in IT : 100
Enter Marks obtained in Cell Biology : 95
Enter Marks obtained in Bioinformatics : 98

*****
FINAL RESULT
*****

Student Name : Elizabeth
Student Roll No. : 115
Student Address : Mumbai
Course Name : MScBioinformatics
Course Code : GNKP BIOINF0101

Marks Obtained in :
    IT : 100
    Cell Biology : 95
    Bioinformatics : 98

Total Marks Obtained : 293 / 300
Percentage Obtained : 97 %
```

Figure 2a: Displaying the student details, course details and marks using multilevel inheritance

RESULTS:

Using the C++ programming language, the concept of inheritance was explored and demonstrated in order to perform single inheritance to calculate the area of a triangle and multilevel inheritance to display the details of a student, details of the course and the marks obtained.

CONCLUSION:

Using the C++ programming language, the concept of inheritance was explored and demonstrated through single inheritance and multilevel inheritance.

REFERENCES:

1. P. (2022, March 31). C++ Inheritance. <https://www.programiz.com/cpp-programming/inheritance>
2. Inheritance in C. (2023, September 6). GeeksforGeeks. <https://www.geeksforgeeks.org/inheritance-in-c/>
3. C++ Inheritance. (n.d.). https://www.w3schools.com/cpp/cpp_inheritance.asp

Practical 9

C++ : Polymorphism, Virtual Function and Friend Function

AIM:

To explore and demonstrate the concepts of polymorphism, virtual function and friend function in C++ programming language.

INTRODUCTION:

C++ is an object-oriented, multi-paradigm language that was developed as an extension of the C programming language and was designed for system and application programming. It supports procedural, functional, and generic programming styles.

Polymorphism is a key concept in object-oriented programming (OOP) that allows a single entity (function or object) to exhibit different behaviours in different contexts. Polymorphism offers several benefits in C++ programming –

1. **Code Reusability** – Polymorphism allows you to write a single function or class that can handle different data types or perform different operations, reducing code duplication and improving code maintainability.
2. **Flexibility** – Polymorphism makes your code more flexible by allowing you to change the behaviour of functions or objects at runtime, based on the specific context.

TYPES OF POLYMORPHISM

1. **Compile – Time Polymorphism –**

Compile – Time Polymorphism can be achieved by function overloading, which is the ability to define multiple functions with the same name but different parameter lists. The compiler determines which function to call based on the specific types of arguments passed to it. This allows you to define a single function name that can handle different types of data or perform different operations.

2. **Runtime Polymorphism –**

Runtime polymorphism, also known as late binding, enables objects of different classes to respond to the same method call in different ways.

(a) **Function Overriding –**

Function overriding allows a derived class to redefine a function from its base class with the same name and signature (return type and parameter list). This allows for different behaviours for the same function call, depending on the object type.

(b) **Virtual Function –**

A virtual function is a member function in a base class that is declared with the **virtual** keyword. This keyword informs the compiler that the implementation

of a function should be determined at runtime based on the actual object type, rather than being statically bound at compile time.

When a virtual function is called through a base class pointer or reference, the compiler determines the actual function implementation to invoke based on the dynamic type of the object being pointed or referenced. This dynamic dispatch mechanism is what enables runtime polymorphism.

Runtime polymorphism allows for more dynamic behaviour but is slower compared to compile – time polymorphism.

FRIEND FUNCTION

A friend function is a function that can access the private and protected members of a class even though it is not a member of that class. Within the class, it is specified using the **friend** keyword. This gives the friend function access to the class's private and protected members, increasing code development efficiency and flexibility while maintaining data encapsulation.

PROGRAMS:

Question No. 1 –

Write a program to perform addition of two and three numbers using method overloading.

CODE:

```
#include<iostream.h>
#include<conio.h>

class Addition
{
    public:
        int sum;

    public:
        void add(int x, int y)
        {
            sum = x + y;
            cout << " Sum of " << x << " and " << y << " = " << sum <<
            endl;
        }

        void add(int a, int b, int c)
        {
            sum = a + b + c;
            cout << " Sum of " << a << ", " << b << " and " << c << " = "
            << sum << endl;
        }
};
```

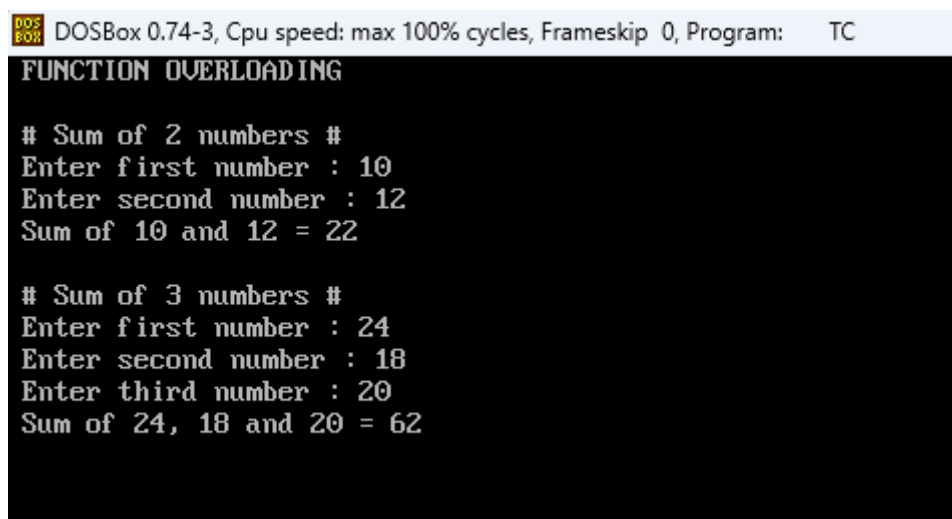
```

void main()
{
    clrscr();
    int a, b, c;
    Addition obj;
    cout << " FUNCTION OVERLOADING" << endl << endl;
    cout << " # Sum of 2 numbers # " << endl;
    cout << " Enter first number : ";
    cin >> a;
    cout << " Enter second number : ";
    cin >> b;
    obj.add(a,b);

    cout << endl;
    cout << " # Sum of 3 numbers # " << endl;
    cout << " Enter first number : ";
    cin >> a;
    cout << " Enter second number : ";
    cin >> b;
    cout << " Enter third number : ";
    cin >> c;
    obj.add(a,b,c);
    getch();
}

```

OUTPUT:



```

DOS BOX DOSBox 0.74-3, Cpu speed: max 100% cycles, Frameskip 0, Program: TC
FUNCTION OVERLOADING
# Sum of 2 numbers #
Enter first number : 10
Enter second number : 12
Sum of 10 and 12 = 22
# Sum of 3 numbers #
Enter first number : 24
Enter second number : 18
Enter third number : 20
Sum of 24, 18 and 20 = 62

```

Figure 1: Displaying the sum of two numbers and three numbers using method overloading

Question No. 2 –

Write a program to demonstrate method overriding using virtual function.

CODE:

```
#include<iostream.h>
#include<conio.h>

class Addition1
{
    public:
        int a, b, sum;

    public:
        virtual void add()
        {
            cout << " # Sum of 2 numbers # " << endl;
            cout << " Enter first number : ";
            cin >> a;
            cout << " Enter second number : ";
            cin >> b;
            sum = a + b;
            cout << " Sum of " << a << " and " << b << " = " << sum <<
            endl << endl;
        }
};

class Addition2 : public Addition1
{
    public:
        int x, y, z, sum;

    public:
        void add()
        {
            cout << " # Sum of 3 numbers # " << endl;
            cout << " Enter first number : ";
            cin >> x;
            cout << " Enter second number : ";
            cin >> y;
            cout << " Enter third number : ";
            cin >> z;

            sum = x + y + z;
```

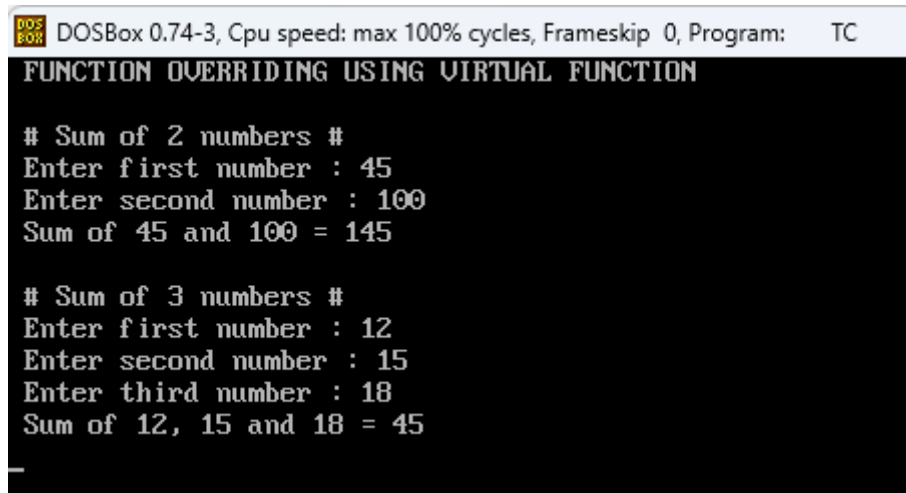
```

        cout << " Sum of " << x << ", " << y << " and " << z << " = "
        << sum << endl;
    }
};

void main()
{
    clrscr();
    cout << " FUNCTION OVERRIDING USING VIRTUAL FUNCTION" <<
    endl << endl;
    Addition1 *ptr;
    Addition1 obj1;
    Addition2 obj2;
    ptr = &obj1;
    ptr -> add();
    ptr = &obj2;
    ptr -> add();
    getch();
}

```

OUTPUT:



```

DOSBox 0.74-3, Cpu speed: max 100% cycles, Frameskip 0, Program: TC
FUNCTION OVERRIDING USING VIRTUAL FUNCTION

# Sum of 2 numbers #
Enter first number : 45
Enter second number : 100
Sum of 45 and 100 = 145

# Sum of 3 numbers #
Enter first number : 12
Enter second number : 15
Enter third number : 18
Sum of 12, 15 and 18 = 45

```

Figure 2: Displaying the sum of two numbers and three numbers by method overriding using virtual function

RESULTS:

Using the C++ programming language, the concepts of polymorphism, virtual function and friend function was explored and demonstrated by performing the summation of two numbers and three numbers using the method overloading technique and demonstrating method overriding using the virtual function.

CONCLUSION:

Using the C++ programming language, the concepts of polymorphism, virtual function and friend function were explore and demonstrated by implementing the techniques of method overloading and method overriding using the virtual function.

REFERENCES:

1. *C Polymorphism*. (2023, November 16). GeeksforGeeks.
<https://www.geeksforgeeks.org/cpp-polymorphism/>
 2. *C++ Polymorphism*. (n.d.). https://www.w3schools.com/cpp/cpp_polymorphism.asp
 3. J. (n.d.). *GitHub - juancsosap/cpptraining*. GitHub.
<https://github.com/juancsosap/cpptraining>
-

Practical 10

C++ : Constructor and Destructor

AIM:

To explore and demonstrate the concept of constructors and destructors in C++ programming language.

INTRODUCTION:

C++ is an object-oriented, multi-paradigm language that was developed as an extension of the C programming language and was designed for system and application programming. It supports procedural, functional, and generic programming styles.

CONSTRUCTOR

A constructor is a member function of a class that has the same name as the class name and does not have any return type, not even void. It is called whenever an instance of the class is created and can be defined with or without arguments.

A constructor aids to initialize the object of a class. It is used to allocate the memory to an object of the class. It can be defined manually with arguments or without arguments. There can be multiple constructors in a class. It can be overloaded but it cannot be inherited or virtual.

General Syntax of Constructor –

```
class class_name // class
{
    // access specifiers
    // data members and member functions

    class_name( ) //constructor
    {
        // code to be executed
    }
};
```

Types of Constructors –

1. **Default constructor** – This constructor is called automatically when an object of a class is created without any arguments. It is used to initialize the data members of the object to default values. The compiler will automatically generate a default constructor if not user – defined.
2. **Parameterized constructor** – This constructor takes one or more arguments and is used to initialize the data members of an object with specific values. You can define as many overloaded constructors as needed to customize initialization in various ways. Parameterized constructors are typically declared as public so that code outside the class definition or inheritance hierarchy can create objects of the class.

3. **Copy constructor** – This constructor creates a new object by initializing it with an existing object of the same class. It is typically used to pass objects by value as function arguments or return values.

DESTRUCTOR

Like a constructor, a Destructor is also a member function of a class that has the same name as the class name preceded by a tilde (~) operator. It helps to deallocate the memory of an object. It is called while the object of the class is freed or deleted. In a class, there is always a single destructor without any parameters so it cannot be overloaded. If a class is inherited by another class and both the classes have a destructor then the destructor of the child class is called first, followed by the destructor of the parent or base class.

General Syntax of Destructor –

```
class class_name // class
{
    // access specifiers
    // data members and member functions

    class_name( ) //constructor
    {
        // code to be executed
    }

    ~class_name( ) //destructor
    {
        //code to be executed
    }
};
```

PROGRAMS:

Question No. 1 –

Write a program to demonstrate Constructor and Destructor.

CODE:

```
#include<iostream.h>
#include<conio.h>

class class1
{
    int num1;

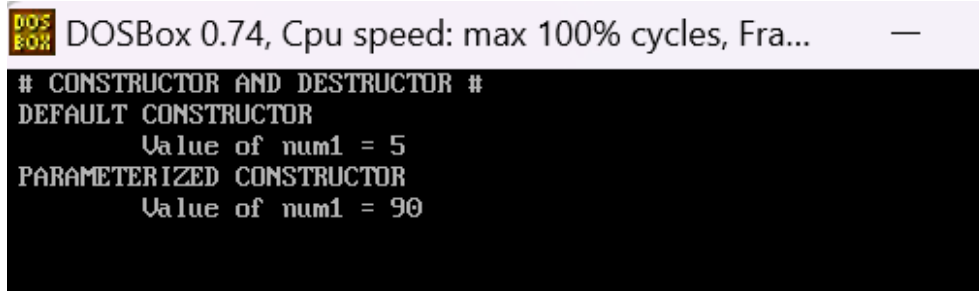
public:
    class1()
    {
        cout << " DEFAULT CONSTRUCTOR" << endl;
        num1 = 5;
        cout << "\t Value of num1 = " << num1 << endl;
    }

    class1(int num2)
    {
        cout << " PARAMETERIZED CONSTRUCTOR" << endl;
        num1 = num2;
        cout << "\t Value of num1 = " << num1 << endl;
    }

    ~class1()
    {
        cout << " DESTRUCTOR EXECUTED" << endl;
    }
};

void main()
{
    clrscr();
    cout << " # CONSTRUCTOR AND DESTRUCTOR #" << endl;
    class1 c1;
    class1 c2(90);
    getch();
}
```

OUTPUT:

A screenshot of a DOSBox window. The title bar reads "DOSBox 0.74, Cpu speed: max 100% cycles, Fra...". The window contains a black terminal area with white text. The text is as follows:

```
# CONSTRUCTOR AND DESTRUCTOR #  
DEFAULT CONSTRUCTOR  
    Value of num1 = 5  
PARAMETERIZED CONSTRUCTOR  
    Value of num1 = 90
```

Figure 1: Executing Constructor and Destructor

RESULTS:

Using the C++ programming language, the concept of constructors and destructors was explored and demonstrated by creating of a default constructor, a parameterized constructor and a destructor to deallocate the memory of the object created.

CONCLUSION:

Using the C++ programming language, the concept of constructors and destructors was explored and demonstrated.

REFERENCES:

1. Rai, R. (2022, September 3). *Constructor and Destructor in C++*. Codementor. <https://www.codementor.io/@supernerdd7/constructor-and-destructor-in-c-1r8kkogm6j>
 2. *Difference Between Constructor and Destructor in C*. (2022, June 23). GeeksforGeeks. <https://www.geeksforgeeks.org/difference-between-constructor-and-destructor-in-c/>
-