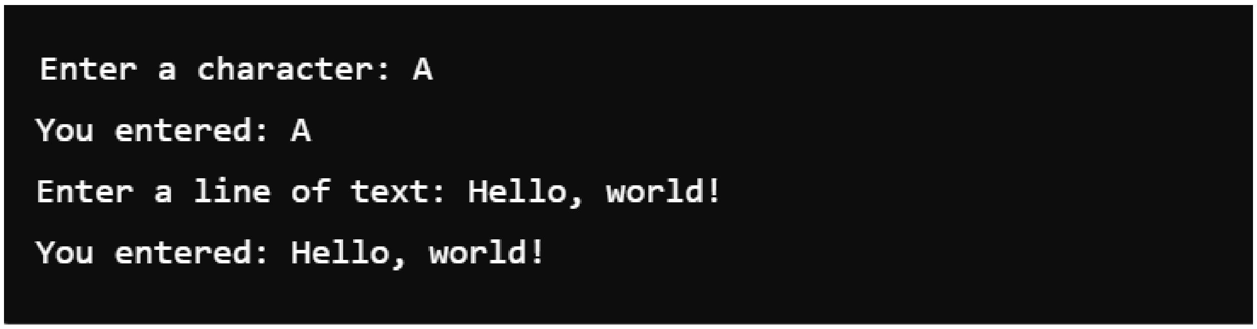
Subject:- C++
LAB Assignment - 10

1)Write a program that shows use of get() and put() function.

```
#include <iostream>
using namespace std;

int main() {
    char ch;

    // Using get() to read a character from standard input
    cout << "Enter a character: ";
    ch = cin.get();  // Read a character using get()

    // Display the character using put()
    cout << "You entered: ";
    cout.put(ch);    55// Write the character using put()
    cout << endl;

    // Using get() to read a line of text
    cin.ignore();  // Ignore the newline character left in the input buffer
    cout << "Enter a line of text: ";

    // Creating a character array to store the line
    char line[100];
    cin.getline(line, 100);  // Use getline to read a line of text

    // Display the line of text
    cout << "You entered: ";
    cout.put(line[0]);  // Display the first character of the line
    for (int i = 1; line[i] != '\0'; i++) {
        cout.put(line[i]);  // Use put() to display each character of the line
    }
    cout << endl;

    return 0;
}
```

output:

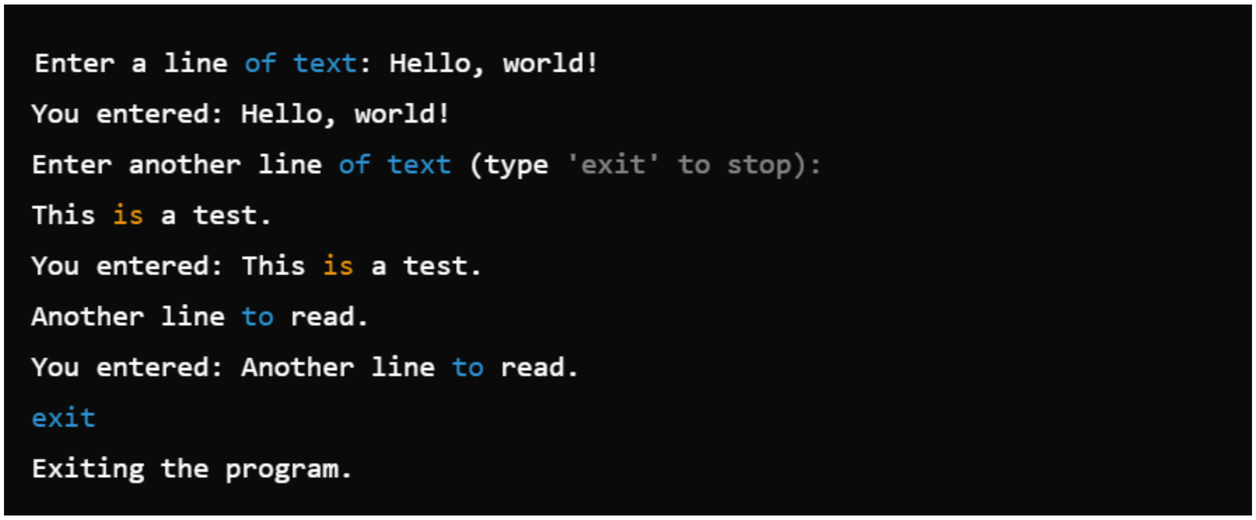


```
Enter a character: A
You entered: A
Enter a line of text: Hello, world!
You entered: Hello, world!
```

2) Write a program that shows use of getline() function.

```
#include <iostream>
#include <string> // Include the string header for std::string
using namespace std;

int main() {
    string input; // Declare a string variable to hold the input

    // Prompt the user to enter a line of text
    cout << "Enter a line of text: ";
    getline(cin, input); // Use getline to read a full line

    // Display the entered line
    cout << "You entered: " << input << endl;
    // Example: Reading multiple lines using getline
    cout << "Enter another line of text (type 'exit' to stop):" << endl;
    while (true) {
        getline(cin, input); // Read the next line of text

        // Check if the user wants to exit
        if (input == "exit") {
            break; // Exit the loop if the user types 'exit'
        }

        // Display the entered line
        cout << "You entered: " << input << endl;
    }

    cout << "Exiting the program." << endl;

    return 0;
}
```

output:

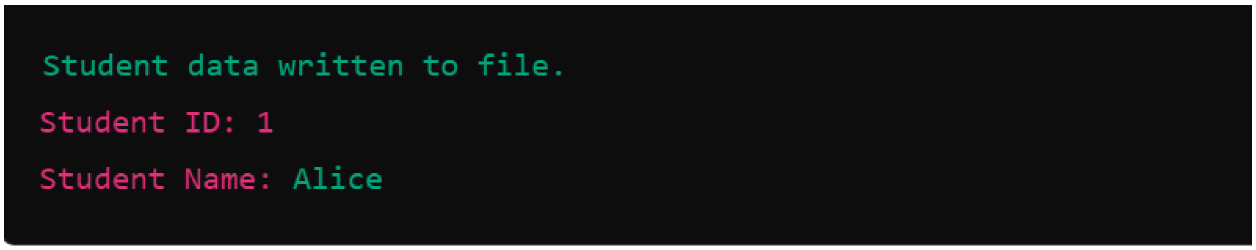


```
Enter a line of text: Hello, world!
You entered: Hello, world!
Enter another line of text (type 'exit' to stop):
This is a test.
You entered: This is a test.
Another line to read.
You entered: Another line to read.
exit
Exiting the program.
```

3) Write a program that shows use of write() function.

```
#include <iostream>
#include <fstream>  // For file operations
using namespace std;

struct Student {
    int id;
    char name[50];
};

int main() {
    // Create a Student object
    Student student1;
    student1.id = 1;
    strcpy(student1.name, "Alice");

    // Write the Student object to a binary file
    ofstream outFile("student.dat", ios::binary); // Open file in binary mode
    if (!outFile) {
        cerr << "Error opening file for writing." << endl;
        return 1;
    }

    // Use write() to write binary data
    outFile.write(reinterpret_cast<char*>(&student1), sizeof(Student));
    outFile.close(); // Close the file

    cout << "Student data written to file." << endl;

    // Read the Student object back from the file
    Student student2;
    ifstream inFile("student.dat", ios::binary); // Open file in binary mode
    if (!inFile) {
        cerr << "Error opening file for reading." << endl;
        return 1;
    }

    // Use read() to read binary data
    inFile.read(reinterpret_cast<char*>(&student2), sizeof(Student));
    inFile.close(); // Close the file

    // Display the read data
    cout << "Student ID: " << student2.id << endl;
    cout << "Student Name: " << student2.name << endl;

    return 0;
}
```

output:



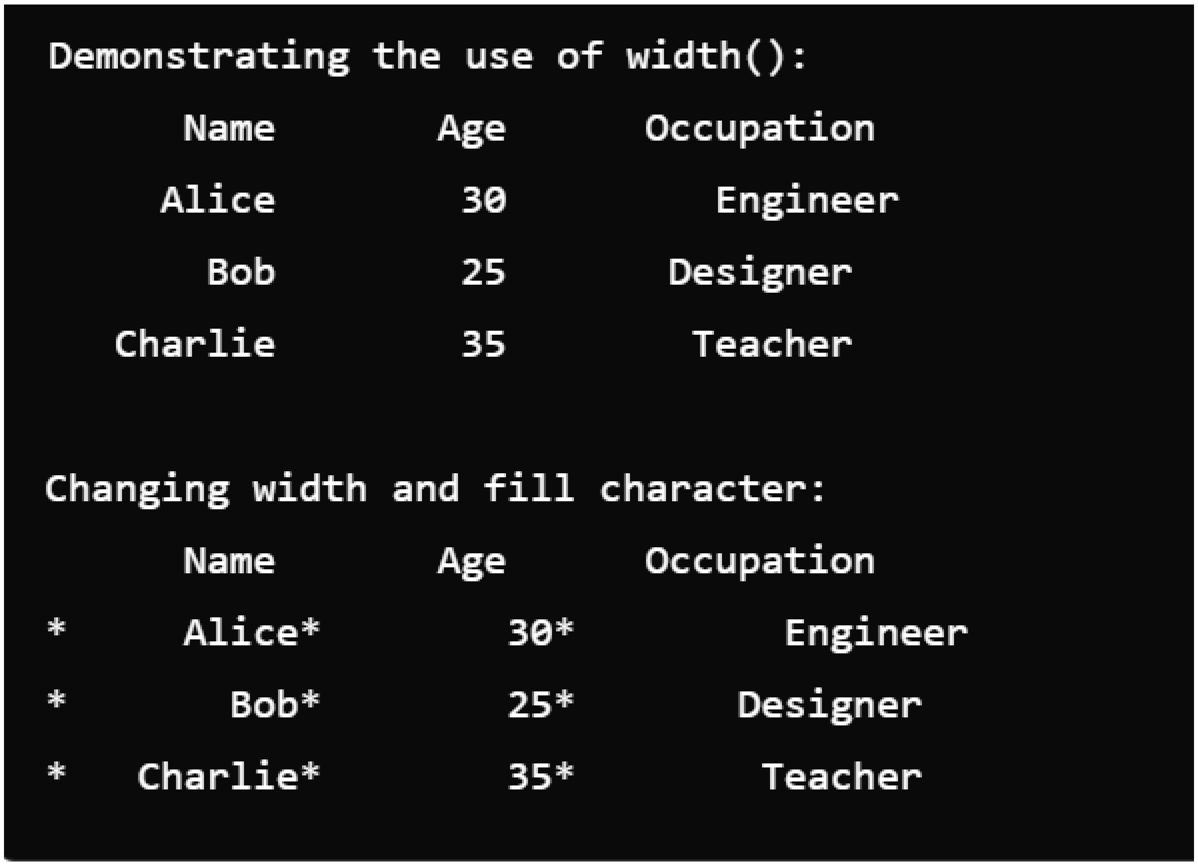
```
Student data written to file.
Student ID: 1
Student Name: Alice
```

4) Write a program that shows use of width().

```
#include <iostream>
#include <iomanip> // Include for std::setw
using namespace std;

int main() {
    cout << "Demonstrating the use of width():\n";

    // Set the width for output
```

```
cout << setw(10) << "Name"
    << setw(10) << "Age"
    << setw(15) << "Occupation" << endl;

// Sample data
cout << setw(10) << "Alice"
    << setw(10) << 30
    << setw(15) << "Engineer" << endl;

cout << setw(10) << "Bob"
    << setw(10) << 25
    << setw(15) << "Designer" << endl;

cout << setw(10) << "Charlie"
    << setw(10) << 35
    << setw(15) << "Teacher" << endl;

// Change the width and fill character
cout << "\nChanging width and fill character:\n";
cout << setfill('*'); // Set fill character to '*'
cout << setw(10) << "Name"
    << setw(10) << "Age"
    << setw(15) << "Occupation" << endl;

cout << setw(10) << "Alice"
    << setw(10) << 30
    << setw(15) << "Engineer" << endl;

cout << setw(10) << "Bob"
    << setw(10) << 25
    << setw(15) << "Designer" << endl;

cout << setw(10) << "Charlie"
    << setw(10) << 35
    << setw(15) << "Teacher" << endl;
    return 0;
}
```

output:

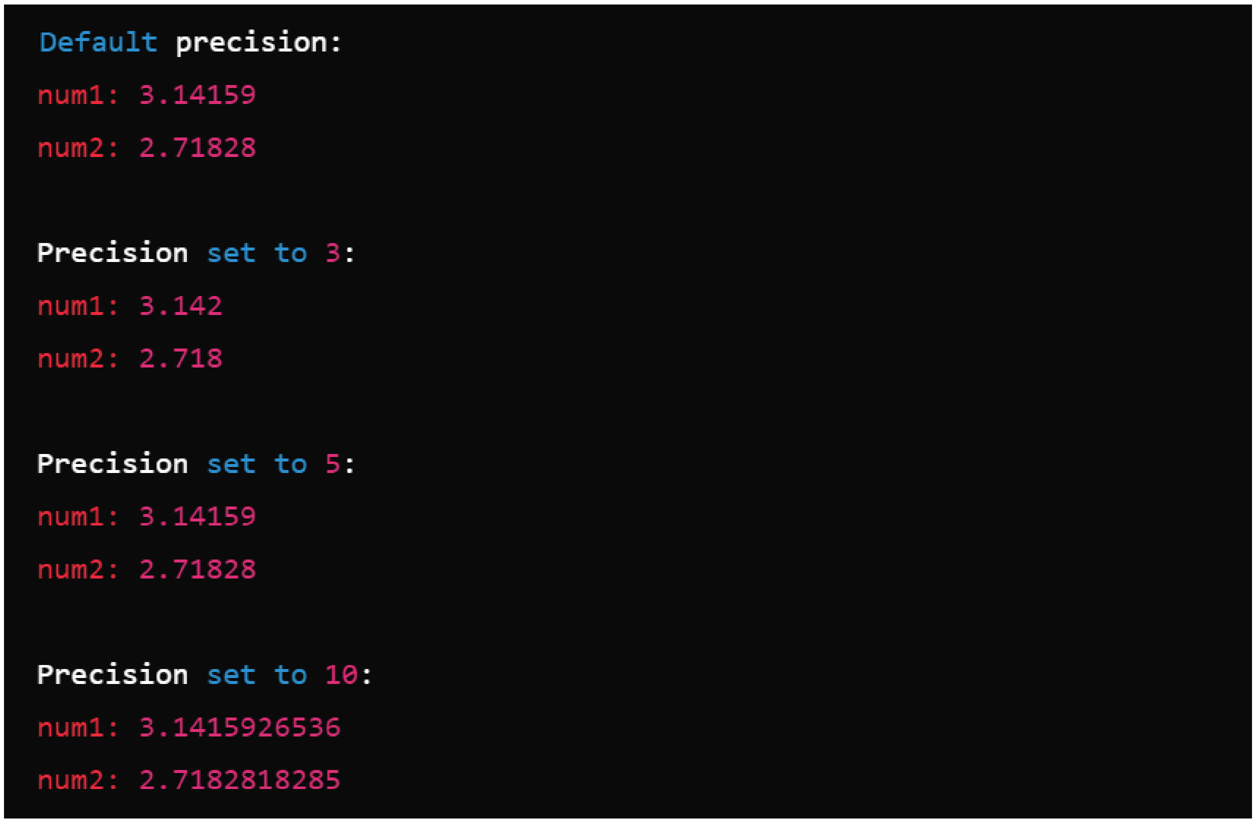


```
Demonstrating the use of width():
      Name      Age     Occupation
     Alice       30        Engineer
       Bob       25      Designer
   Charlie       35        Teacher

Changing width and fill character:
      Name      Age     Occupation
*     Alice*       30*        Engineer
*       Bob*       25*      Designer
*   Charlie*       35*        Teacher
```

5) Write a program that shows use of precision().

```
#include <iostream>
#include <iomanip> // Include for std::setprecision
using namespace std;

int main() {
    double num1 = 3.14159265358979;
    double num2 = 2.71828182845904;
    // Default precision
    cout << "Default precision:\n";
    cout << "num1: " << num1 << endl; // Default precision (usually 6)
    cout << "num2: " << num2 << endl; // Default precision (usually 6)

    // Set precision to 3
    cout << "\nPrecision set to 3:\n";
    cout << fixed << setprecision(3); // Fixed-point notation
    cout << "num1: " << num1 << endl;
    cout << "num2: " << num2 << endl;

    // Set precision to 5
    cout << "\nPrecision set to 5:\n";
    cout << setprecision(5); // Applies to fixed-point as well
    cout << "num1: " << num1 << endl;
    cout << "num2: " << num2 << endl;

    // Set precision to 10
    cout << "\nPrecision set to 10:\n";
    cout << setprecision(10); // Applies to fixed-point as well
    cout << "num1: " << num1 << endl;
    cout << "num2: " << num2 << endl;

    return 0;
}
```

output:



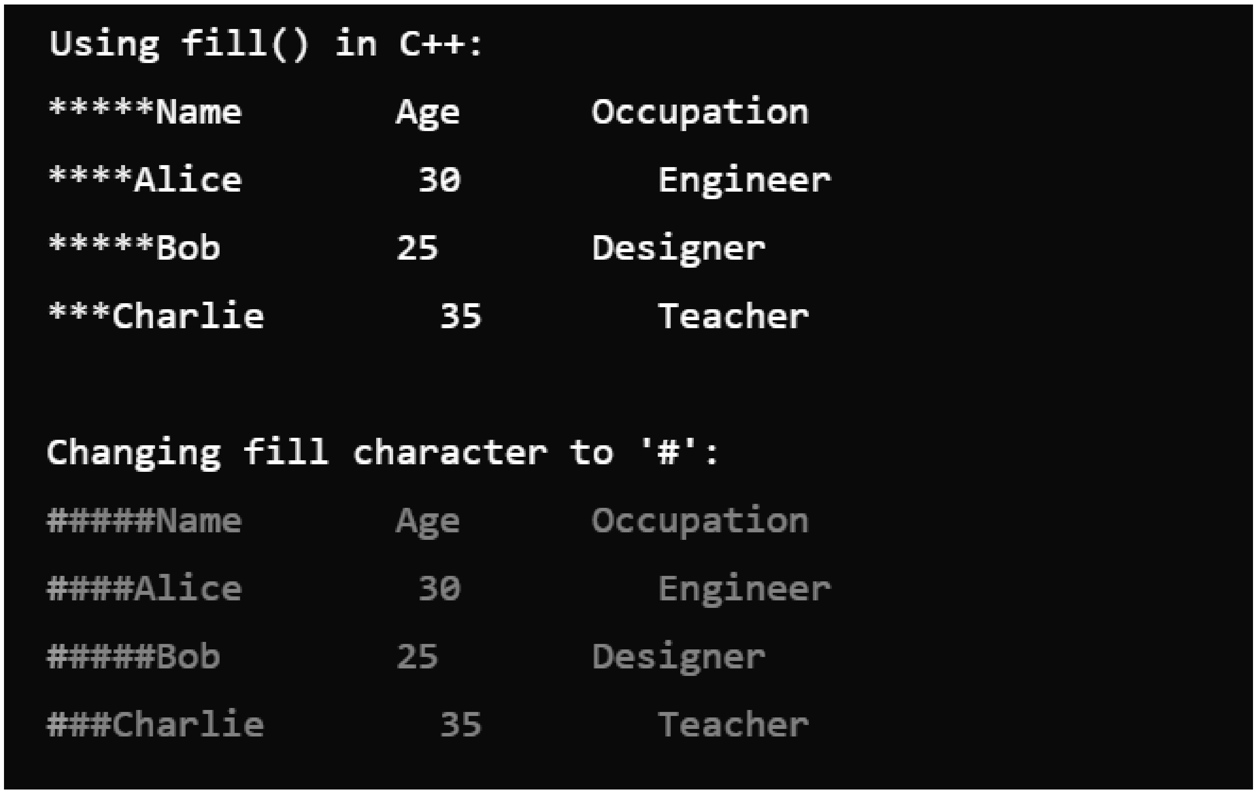
```
Default precision:
num1: 3.14159
num2: 2.71828

Precision set to 3:
num1: 3.142
num2: 2.718

Precision set to 5:
num1: 3.14159
num2: 2.71828

Precision set to 10:
num1: 3.1415926536
num2: 2.7182818285
```

6) Write a program that shows use of fill().

```
#include <iostream>
#include <iomanip> // Include for std::setw and std::setfill
using namespace std;

int main() {
    // Display a header
    cout << "Using fill() in C++:\n";

    // Set the fill character to '*'
    cout << setfill('*');

    // Print formatted output using setw()
    cout << setw(10) << "Name"
         << setw(10) << "Age"
```

```
        << setw(15) << "Occupation" << endl;

    // Sample data
    cout << setw(10) << "Alice"
        << setw(10) << 30
        << setw(15) << "Engineer" << endl;

    cout << setw(10) << "Bob"
        << setw(10) << 25
        << setw(15) << "Designer" << endl;

    cout << setw(10) << "Charlie"
        << setw(10) << 35
        << setw(15) << "Teacher" << endl;

    // Change the fill character to '#'
    cout << "\nChanging fill character to '#':\n";
    cout << setfill('#'); // Set fill character to '#'

    // Print formatted output using setw() with new fill character
    cout << setw(10) << "Name"
        << setw(10) << "Age"
        << setw(15) << "Occupation" << endl;

    cout << setw(10) << "Alice"
        << setw(10) << 30
        << setw(15) << "Engineer" << endl;

    cout << setw(10) << "Bob"
        << setw(10) << 25
        << setw(15) << "Designer" << endl;

    cout << setw(10) << "Charlie"
        << setw(10) << 35
        << setw(15) << "Teacher" << endl;
    return 0;
}
```

output:



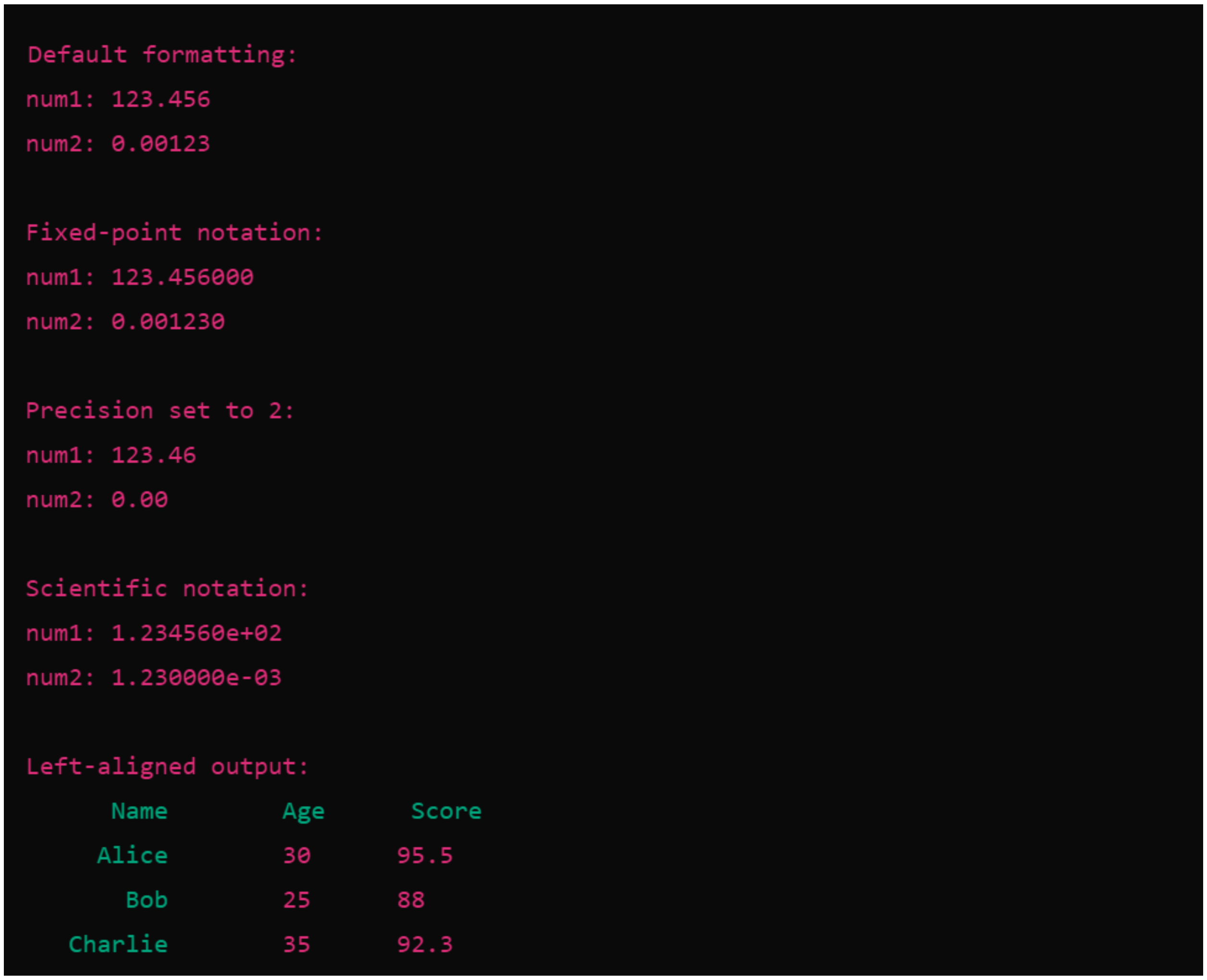
```
Using fill() in C++:
*****Name       Age      Occupation
****Alice        30          Engineer
*****Bob        25       Designer
***Charlie        35         Teacher

Changing fill character to '#':
#####Name       Age      Occupation
####Alice        30          Engineer
#####Bob        25       Designer
###Charlie        35         Teacher
```

7) Write a program that shows formatting with FLAGS in setf().

```
#include <iostream>
#include <iomanip> // Include for std::setf and std::fixed
using namespace std;

int main() {
    double num1 = 123.456;
    double num2 = 0.00123;
    // Default output
    cout << "Default formatting:\n";
    cout << "num1: " << num1 << "\nnum2: " << num2 << endl;

    // Set fixed-point notation
    cout.setf(ios::fixed); // Set the fixed flag
    cout << "\nFixed-point notation:\n";
    cout << "num1: " << num1 << "\nnum2: " << num2 << endl;

    // Set precision to 2
    cout.precision(2);
    cout << "\nPrecision set to 2:\n";
    cout << "num1: " << num1 << "\nnum2: " << num2 << endl;

    // Set scientific notation
    cout.setf(ios::scientific); // Set the scientific flag
    cout << "\nScientific notation:\n";
    cout << "num1: " << num1 << "\nnum2: " << num2 << endl;

    // Reset to default precision
    cout.precision(6);
    cout.setf(ios::fixed); // Use fixed-point again for the next example

    // Set the left alignment flag
    cout.setf(ios::left, ios::adjustfield); // Align output to the left
    cout << "\nLeft-aligned output:\n";
    cout << setw(10) << "Name" << setw(10) << "Age" << setw(10) <<
"Score" << endl;
    cout << setw(10) << "Alice" << setw(10) << 30 << setw(10) << 95.5 <<
endl;
    cout << setw(10) << "Bob" << setw(10) << 25 << setw(10) << 88.0 <<
endl;
    cout << setw(10) << "Charlie" << setw(10) << 35 << setw(10) << 92.3 <<
endl;
```

```
    return 0;
}
```

output:

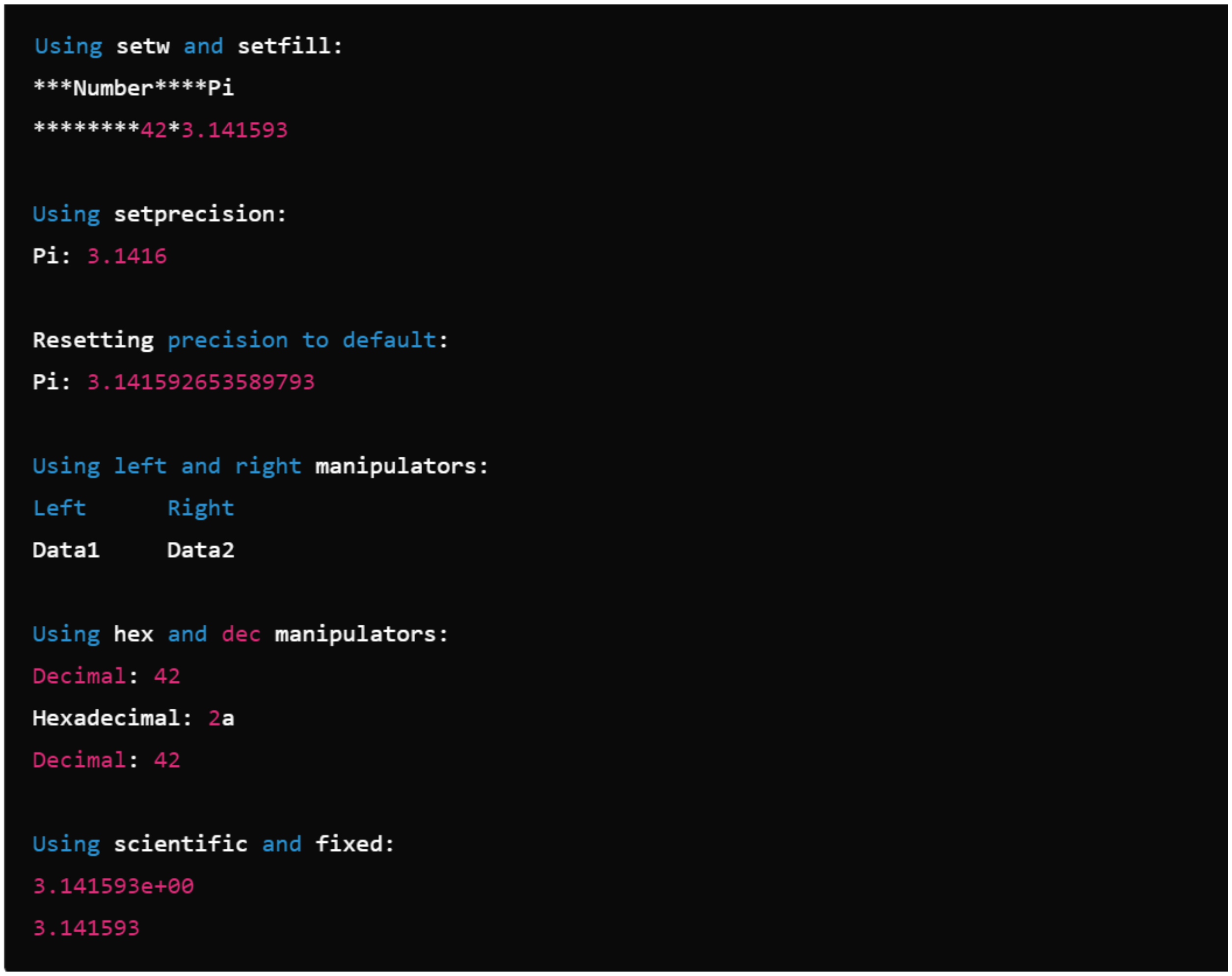


```
Default formatting:
num1: 123.456
num2: 0.00123

Fixed-point notation:
num1: 123.456000
num2: 0.001230

Precision set to 2:
num1: 123.46
num2: 0.00

Scientific notation:
num1: 1.234560e+02
num2: 1.230000e-03

Left-aligned output:
      Name          Age       Score
     Alice          30       95.5
       Bob          25       88
   Charlie          35       92.3
```

8) Write a program that shows use of different manipulators.

```
#include <iostream>
#include <iomanip> // Include for manipulators like setw, setfill, setprecision
using namespace std;

int main() {
    double pi = 3.141592653589793;
    int number = 42;

    // Using setw and setfill
    cout << "Using setw and setfill:\n";
    cout << setfill('*'); // Set fill character to '*'
    cout << setw(10) << "Number"
        << setw(10) << "Pi" << endl;
    cout << setw(10) << number
        << setw(10) << pi << endl;

    // Using setprecision
    cout << "\nUsing setprecision:\n";
    cout << fixed << setprecision(4); // Set precision to 4 decimal places
    cout << "Pi: " << pi << endl;

    // Reset precision to default
    cout << "\nResetting precision to default:\n";
    cout << defaultfloat; // Reset to default float format
    cout << "Pi: " << pi << endl;

    // Using left and right manipulators
    cout << "\nUsing left and right manipulators:\n";
    cout << left << setw(10) << "Left"
        << right << setw(10) << "Right" << endl;

    cout << left << setw(10) << "Data1"
        << right << setw(10) << "Data2" << endl;

    // Using hex and dec
    cout << "\nUsing hex and dec manipulators:\n";
    cout << "Decimal: " << number << endl;
    cout << "Hexadecimal: " << hex << number << endl; // Convert to
hexadecimal
    cout << dec << number << endl; // Convert back to decimal
    // Using scientific and fixed
    cout << "\nUsing scientific and fixed:\n";
    cout << scientific << pi << endl; // Scientific notation
    cout << fixed << pi << endl; // Fixed-point notation

    return 0;
}
```

output:

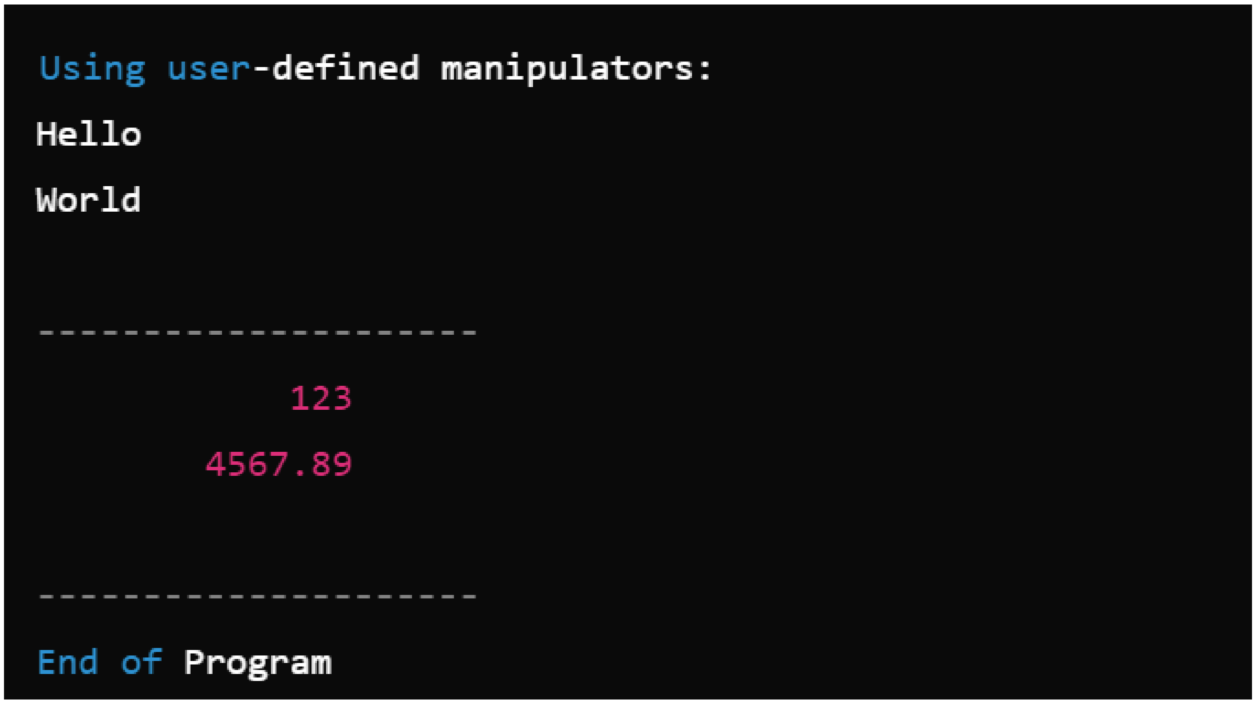


```
Using setw and setfill:
***Number****Pi
********42*3.141593

Using setprecision:
Pi: 3.1416

Resetting precision to default:
Pi: 3.141592653589793

Using left and right manipulators:
Left      Right
Data1     Data2

Using hex and dec manipulators:
Decimal: 42
Hexadecimal: 2a
Decimal: 42

Using scientific and fixed:
3.141593e+00
3.141593
```

9) Write a program of user defined manipulator.

```
#include <iostream>
#include <iomanip> // Include for std::setw
using namespace std;

// User-defined manipulator to set a custom fill character and width
ostream& customFill(ostream& os) {
    os << setfill('-') << setw(15);
    return os;
}

// User-defined manipulator to print a border
ostream& border(ostream& os) {
    os << "\n--------------------";
    return os;
}

int main() {
    cout << "Using user-defined manipulators:\n";

    // Using customFill manipulator
    cout << customFill << "Hello" << endl;  // Custom fill with width
    cout << customFill << "World" << endl;  // Custom fill with width

    // Using border manipulator
    cout << border << endl;

    // Displaying some numbers with customFill
    cout << customFill << 123 << endl; // Custom fill for an integer
    cout << customFill << 4567.89 << endl; // Custom fill for a floating-point
number

    cout << border << endl; // Another border
    // Demonstrating multiple uses
    cout << customFill << "End of Program" << endl;

    return 0;
}
```

output:

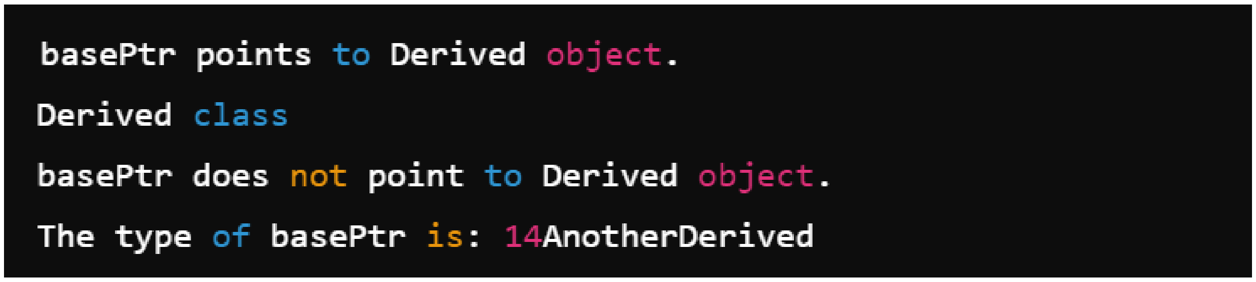


```
Using user-defined manipulators:
Hello
World


---------------------
            123
        4567.89


---------------------
End of Program
```

10) Write a program for RTTI.

```
#include <iostream>
#include <typeinfo> // Include for typeid
using namespace std;

// Base class
class Base {
public:
    virtual void show() {
        cout << "Base class" << endl;
    }
    virtual ~Base() {} // Virtual destructor
};

// Derived class
class Derived : public Base {
public:
    void show() override {
        cout << "Derived class" << endl;
    }
};

// Another derived class
class AnotherDerived : public Base {
public:
    void show() override {
        cout << "AnotherDerived class" << endl;
    }
};

int main() {
    Base* basePtr; // Base class pointer

    // Create a Derived object
    Derived derivedObj;
    basePtr = &derivedObj; // Point to Derived object

    // Using dynamic_cast to check the type at runtime
    if (Derived* d = dynamic_cast<Derived*>(basePtr)) {
        cout << "basePtr points to Derived object." << endl;
        d->show(); // Call Derived's show()
    } else {
        cout << "basePtr does not point to Derived object." << endl;
    }

    // Create another Derived object
    AnotherDerived anotherDerivedObj;
    basePtr = &anotherDerivedObj; // Point to AnotherDerived object

    // Using dynamic_cast to check the type again
    if (Derived* d = dynamic_cast<Derived*>(basePtr)) {
        cout << "basePtr points to Derived object." << endl;
        d->show(); // Call Derived's show()
    } else {
        cout << "basePtr does not point to Derived object." << endl;
    }

    // Using typeid to get the actual type of the object
    cout << "The type of basePtr is: " << typeid(*basePtr).name() << endl;

    return 0;
}
```

Output:



```
basePtr points to Derived object.
Derived class
basePtr does not point to Derived object.
The type of basePtr is: 14AnotherDerived
```