

Unit 1: Fundamentals of C++

Short Answer Questions:

Q. Write the syntax of writing comments in C++. (Nov 20)

Ans. In C++, single-line comments use `//`, and multi-line comments are written using `/* comment */`.

C++ ਵਿੱਚ ਇੱਕ-ਲਾਈਨ ਟਿੱਪਣੀ ਲਈ `//` ਵਰਤੀ ਜਾਂਦੀ ਹੈ ਅਤੇ ਬਹੁ-ਲਾਈਨ ਟਿੱਪਣੀਆਂ `/*` ਟਿੱਪਣੀ `*/` ਰੂਪ ਵਿੱਚ ਲਿਖੀਆਂ ਜਾਂਦੀਆਂ ਹਨ।

Q. List various types of operators and their usage. (Nov 22)

Ans. C++ includes arithmetic, relational, logical, assignment, increment/decrement, and bitwise operators used for performing various computations.

C++ ਵਿੱਚ ਗਣਿਤਕ, ਰਿਸ਼ਤਾਕਾਰਕ, ਤਰਕਸੰਗਤ, ਨਿਰਧਾਰਣ, ਵਾਧੂ/ਘਟਾਓ, ਅਤੇ ਬਿੱਟਵਾਈਜ਼ ਓਪਰੇਟਰ ਸ਼ਾਮਲ ਹੁੰਦੇ ਹਨ, ਜੋ ਵੱਖ-ਵੱਖ ਗਣਨਾਵਾਂ ਕਰਨ ਲਈ ਵਰਤੇ ਜਾਂਦੇ ਹਨ।

Q. Data types (Nov 24)

Ans. Data types in C++ define the type of data a variable can hold, such as int, float, char, double, and bool.

C++ ਵਿੱਚ ਡਾਟਾ ਟਾਈਪ ਇਹ ਨਿਰਧਾਰਤ ਕਰਦਾ ਹੈ ਕਿ ਕੋਈ ਵੀ ਵੇਰੀਏਬਲ ਕਿਸ ਕਿਸਮ ਦਾ ਡਾਟਾ ਰੱਖ ਸਕਦਾ ਹੈ, ਜਿਵੇਂ ਕਿ int, float, char, double, ਅਤੇ bool।

Q. Continue (Nov 24)

Ans. The continue statement is commonly used in for, while, or do-while loops to bypass remaining code and re-evaluate the loop condition.

continue ਸਟੇਟਮੈਂਟ ਆਮ ਤੌਰ 'ਤੇ for, while, ਜਾਂ do-while ਲੂਪ ਵਿੱਚ ਵਰਤੀ ਜਾਂਦੀ ਹੈ, ਤਾਕਿ ਬਾਕੀ ਕੋਡ ਨੂੰ ਛੱਡ ਕੇ ਲੂਪ ਦੀ ਸ਼ਰਤ ਨੂੰ ਮੁੜ ਜਾਂਚਿਆ ਜਾ ਸਕੇ।

Q. When do you need to use continue statement? (Nov 22)

Ans. The continue statement is used inside loops to skip the current iteration and jump to the next cycle. continue ਸਟੇਟਮੈਂਟ ਲੂਪ ਦੇ ਅੰਦਰ ਵਰਤੀ ਜਾਂਦੀ ਹੈ ਤਾਂ ਜੋ ਮੌਜੂਦਾ ਇਟਰੇਸ਼ਨ ਨੂੰ ਛੱਡ ਕੇ ਲੂਪ ਦੀ ਅਗਲੀ ਚੱਕਰ ਤੇ ਜਾਏ।

Q. What is need of type casting in C++? (Nov 22)

Ans. Type casting is used to convert one data type into another to ensure correct computations and avoid data loss or mismatch.

Type casting ਦੀ ਵਰਤੋਂ ਇੱਕ ਡਾਟਾ ਟਾਈਪ ਨੂੰ ਦੂਜੇ ਵਿੱਚ ਬਦਲਣ ਲਈ ਕੀਤੀ ਜਾਂਦੀ ਹੈ ਤਾਂ ਜੋ ਸਹੀ ਗਣਨਾ ਹੋ ਸਕੇ ਅਤੇ ਡਾਟਾ ਦੀ ਗੁੰਮਸ਼ੁਦਾ ਜਾਂ ਗਲਤ ਮੈਚਿੰਗ ਤੋਂ ਬਚਿਆ ਜਾ ਸਕੇ।

Long Answer Questions:

Q. What are operators in C++? Explain the types of operators. How is the precedence and associativity of operators determined? (Nov 20)

Ans. **Operators** in C++ are special symbols or keywords used to perform operations on variables and values. They are essential for performing computations, making decisions, and manipulating data.

Types of Operators:

1. **Arithmetic Operators:** Used for basic mathematical operations: `+`, `-`, `*`, `/`, `%`
Example: `a + b`, `x % y`
2. **Relational Operators:** Compare values and return true/false: `==`, `!=`, `>`, `<`, `>=`, `<=`
Example: `a > b`

3. **Logical Operators:** Used in decision-making: && (AND), || (OR), ! (NOT)
Example: (a > b && b < c)
4. **Assignment Operators:** Assign values: =, +=, -=, *=, /=, %=
Example: x += 5 (same as x = x + 5)
5. **Increment/Decrement Operators:** Increase or decrease value: ++, --
Example: i++, --j
6. **Bitwise Operators:** Operate on binary representations: &, |, ^, ~, <<, >>
7. **Ternary Operator:** A shorthand for if-else: condition ? expr1 : expr2

Precedence and Associativity: Operator **precedence** determines which operator is evaluated first in an expression. **Associativity** defines the direction (left-to-right or right-to-left) in which operators with the same precedence are evaluated. For example, * and / have higher precedence than + and -, and most operators are left-associative.

C++ ਵਿੱਚ Operators ਖਾਸ symbols ਜਾਂ keywords ਹੁੰਦੇ ਹਨ ਜੋ variables ਅਤੇ values 'ਤੇ operations ਕਰਨ ਲਈ ਵਰਤੇ ਜਾਂਦੇ ਹਨ। ਇਹ operations ਕਰਨ, ਫੈਸਲੇ ਲੈਣ ਅਤੇ ਡਾਟਾ ਨੂੰ ਮੋਡੀਫਾਈ ਕਰਨ ਵਿੱਚ ਜ਼ਰੂਰੀ ਹੁੰਦੇ ਹਨ।

Operators ਦੇ ਕਿਸਮਾਂ:

1. **Arithmetic Operators (ਗਣਿਤੀ ਓਪਰੇਟਰ):** ਬੁਨਿਆਦੀ ਗਣਿਤੀ ਕਰਵਾਈਆਂ ਲਈ ਵਰਤੇ ਜਾਂਦੇ ਹਨ: +, -, *, /, %

ਉਦਾਹਰਨ: a + b, x % y

2. **Relational Operators (ਤੁਲਨਾਤਮਕ ਓਪਰੇਟਰ):** ਮੁੱਲਾਂ ਦੀ ਤੁਲਨਾ ਕਰਦੇ ਹਨ ਅਤੇ true ਜਾਂ false ਰਿਟਰਨ ਕਰਦੇ ਹਨ:

==, !=, >, <, >=, <=

ਉਦਾਹਰਨ: a > b

3. **Logical Operators (ਤਾਰਕਿਕ ਓਪਰੇਟਰ):** ਫੈਸਲੇ ਲੈਣ ਵਾਲੇ expressions ਵਿੱਚ ਵਰਤੇ ਜਾਂਦੇ ਹਨ: && (AND), || (OR), ! (NOT)

ਉਦਾਹਰਨ: (a > b && b < c)

4. **Assignment Operators (ਮੂਲ ਨਿਰਧਾਰਕ ਓਪਰੇਟਰ):** ਮੁੱਲ ਦਿੱਤਾ ਜਾਂ ਅੱਪਡੇਟ ਕੀਤਾ ਜਾਂਦਾ ਹੈ: =, +=, -=, *=, /=, %=

ਉਦਾਹਰਨ: x += 5 (ਇਹ x = x + 5 ਦੇ ਬਰਾਬਰ ਹੈ)

5. **Increment/Decrement Operators (ਵਾਧੂ/ਘਟਾਊ ਓਪਰੇਟਰ):** ਮੁੱਲ ਨੂੰ 1 ਨਾਲ ਵਧਾਉਂਦੇ ਜਾਂ ਘਟਾਉਂਦੇ ਹਨ: ++, --

ਉਦਾਹਰਨ: i++, --j

6. **Bitwise Operators (ਬਿਨਾਰੀ ਓਪਰੇਟਰ):** ਬਿਨਾਰੀ ਰੂਪ ਵਿੱਚ operations ਕਰਦੇ ਹਨ: &, |, ^, ~, <<, >>

7. **Ternary Operator (ਤਿਉਂਕੜੀ ਓਪਰੇਟਰ):** if-else ਦਾ ਛੋਟਾ ਰੂਪ: condition ? expr1 : expr2

Precedence ਅਤੇ Associativity:

- **Precedence (ਤਰਜੀਹ)** ਦੱਸਦੀ ਹੈ ਕਿ ਕਿਹੜਾ operator ਪਹਿਲਾਂ ਚੱਲੇਗਾ।
- **Associativity (ਸੰਯੋਗਤਾ)** ਦੱਸਦੀ ਹੈ ਕਿ ਇੱਕੋ ਪ੍ਰਧਾਨਤਾ ਵਾਲੇ operators ਕਿਸ ਦਿਸ਼ਾ ਵਿੱਚ evaluate ਹੋਣਗੇ — left-to-right ਜਾਂ right-to-left।

ਉਦਾਹਰਨ: * ਅਤੇ / ਦੀ precedence + ਅਤੇ - ਨਾਲੋਂ ਵੱਧ ਹੁੰਦੀ ਹੈ। ਜ਼ਿਆਦਾਤਰ operators left-associative ਹੁੰਦੇ ਹਨ।

Q. Explain the stages of program execution in C/C++. (Nov 20)

Ans. Program execution in C/C++ involves several stages, from writing code to running the final executable. These stages ensure that source code is translated into machine-understandable form and executed correctly.

1. **Editing (Writing the Program):** The first stage is writing the source code in a text editor or IDE (like Code::Blocks, Turbo C++, or Visual Studio). The code is saved with extensions .c or .cpp.
2. **Preprocessing:** The preprocessor processes all **preprocessor directives** like #include, #define, etc., before actual compilation. It includes header files and expands macros.
3. **Compilation:** The compiler translates the preprocessed code into **assembly code** or **object code** (.obj or .o files). It checks for syntax errors during this stage.
4. **Assembly:** The assembler converts the assembly code into **machine code**, generating object files that contain low-level binary instructions.
5. **Linking:** The linker combines one or more object files with required libraries (like stdio.h, iostream) to produce a single executable file (.exe). It resolves external references like functions or variables declared in other files.
6. **Loading and Execution:** The operating system loads the executable file into memory and starts execution. The program runs and interacts with system resources as needed.

These stages ensure the transformation of human-readable code into executable programs efficiently and systematically.

C/C++ ਵਿੱਚ Program Execution ਦੀ ਪ੍ਰਕਿਰਿਆ ਕਈ ਪੜਾਅਾਂ ਵਿੱਚ ਹੁੰਦੀ ਹੈ, ਜਿਸਦਾ ਉਦੇਸ਼ ਸੋਰਸ ਕੋਡ ਨੂੰ ਮਸ਼ੀਨ ਦੁਆਰਾ ਸਮਝਣਯੋਗ ਰੂਪ ਵਿੱਚ ਤਬਦੀਲ ਕਰਨਾ ਅਤੇ ਉਸਨੂੰ ਸਹੀ ਢੰਗ ਨਾਲ ਚਲਾਉਣਾ ਹੁੰਦਾ ਹੈ।

1. **Editing (ਪ੍ਰੋਗ੍ਰਾਮ ਲਿਖਣਾ):** ਸਭ ਤੋਂ ਪਹਿਲਾਂ, ਕੋਡ ਨੂੰ ਕਿਸੇ text editor ਜਾਂ IDE (ਜਿਵੇਂ Code::Blocks, Turbo C++, ਜਾਂ Visual Studio) ਵਿੱਚ ਲਿਖਿਆ ਜਾਂਦਾ ਹੈ। ਇਹ ਸੋਰਸ ਕੋਡ .c ਜਾਂ .cpp extension ਨਾਲ ਸੰਭਾਲਿਆ ਜਾਂਦਾ ਹੈ।

2. **Preprocessing (ਪ੍ਰੀ-ਪ੍ਰੋਸੈਸਿੰਗ):** Preprocessor ਸਾਰੇ preprocessor ਡਾਇਰੈਕਟਿਵਸ ਨੂੰ ਪ੍ਰੋਸੈਸ ਕਰਦਾ ਹੈ, ਜਿਵੇਂ #include, #define, ਆਦਿ। ਇਹ header files ਨੂੰ ਸ਼ਾਮਲ ਕਰਦਾ ਹੈ ਅਤੇ macros ਨੂੰ ਵਿਸਥਾਰਿਤ ਕਰਦਾ ਹੈ।

3. **Compilation (ਕੰਪਾਈਲੇਸ਼ਨ):** Compiler preprocessed ਕੋਡ ਨੂੰ assembly code ਜਾਂ object code ਵਿੱਚ ਤਬਦੀਲ ਕਰਦਾ ਹੈ (.obj ਜਾਂ .o ਫਾਈਲਾਂ)। ਇਹ ਪੜਾਅ syntax errors ਨੂੰ ਵੀ ਜਾਂਚਦਾ ਹੈ।

4. **Assembly (ਅਸੈਂਬਲੀ):** Assembler assembly code ਨੂੰ machine code ਵਿੱਚ ਤਬਦੀਲ ਕਰਦਾ ਹੈ, ਜਿਸ ਨਾਲ object files ਬਣਦੀਆਂ ਹਨ ਜੋ ਕਿ low-level binary instructions ਰੱਖਦੀਆਂ ਹਨ।

5. **Linking (ਲਿੰਕਿੰਗ):** Linker ਇੱਕ ਜਾਂ ਵੱਧ object files ਨੂੰ libraries (ਜਿਵੇਂ stdio.h, iostream) ਨਾਲ ਮਿਲਾ ਕੇ ਇੱਕ executable file ਬਣਾਉਂਦਾ ਹੈ (.exe)। ਇਹ ਉਹਨਾਂ functions ਜਾਂ variables ਦੀ definition ਲੱਭਦਾ ਹੈ ਜੋ ਕਿਸੇ ਹੋਰ file ਵਿੱਚ ਹਨ।

6. **Loading and Execution (ਲੋਡਿੰਗ ਅਤੇ ਚਲਾਉਣਾ):** Operating System executable file ਨੂੰ memory ਵਿੱਚ ਲੋਡ ਕਰਦਾ ਹੈ ਅਤੇ execution ਸ਼ੁਰੂ ਕਰਦਾ ਹੈ। ਹੁਣ ਪ੍ਰੋਗ੍ਰਾਮ system resources ਨਾਲ ਸੰਪਰਕ ਕਰਕੇ ਚੱਲਦਾ ਹੈ।

ਸੰਖੇਪ ਵਿੱਚ, ਇਹ ਸਾਰੇ ਪੜਾਅ ਇਹ ਯਕੀਨੀ ਬਣਾਉਂਦੇ ਹਨ ਕਿ ਮਨੁੱਖੀ ਭਾਸ਼ਾ ਵਿੱਚ ਲਿਖਿਆ ਕੋਡ ਸਹੀ ਤਰੀਕੇ ਨਾਲ ਮਸ਼ੀਨ ਦੁਆਰਾ ਚਲਾਇਆ ਜਾ ਸਕੇ।

Q. What are the various Compound Assignment Operators in C++? State the difference between Pre and Post Increment/Decrement Operations. (Nov 22)

Ans. **Compound Assignment Operators** in C++ are shorthand notations that combine an arithmetic or bitwise operation with assignment. They simplify code and improve readability. The main compound assignment operators include:

- `+=` → Adds and assigns: `a += 5;` is the same as `a = a + 5;`
- `-=` → Subtracts and assigns: `a -= 3;`
- `*=` → Multiplies and assigns: `a *= 2;`
- `/=` → Divides and assigns: `a /= 4;`
- `%=` → Modulus and assigns: `a %= 2;`
- `&=`, `|=`, `^=`, `<<=`, `>>=` → Bitwise assignment operators

These operators help reduce redundancy in code.

Difference Between Pre and Post Increment/Decrement:

- **Pre-Increment/Decrement (`++i`, `--i`):** The variable is incremented or decremented **before** its value is used in an expression.

Example:

```
int a = 5, b;
```

```
b = ++a; // a becomes 6, then b = 6
```

- **Post-Increment/Decrement (`i++`, `i--`):** The original value of the variable is used in the expression **before** it is changed.

Example:

```
int a = 5, b;
```

```
b = a++; // b = 5, then a becomes 6
```

Pre and post operations are often used in loops and expressions requiring controlled value updates.

C++ ਵਿੱਚ Compound Assignment Operators:

Compound Assignment Operators ਛੋਟਾ ਲਿਖਣ ਵਾਲਾ ਤਰੀਕਾ ਹੁੰਦੇ ਹਨ ਜੋ ਕਿ arithmetic ਜਾਂ bitwise operation ਨੂੰ assignment ਦੇ ਨਾਲ ਜੋੜਦੇ ਹਨ। ਇਹ ਕੋਡ ਨੂੰ ਸਰਲ ਅਤੇ ਪੜ੍ਹਨ ਯੋਗ ਬਣਾਉਂਦੇ ਹਨ।

ਮੁੱਖ compound assignment operators ਵਿੱਚ ਸ਼ਾਮਲ ਹਨ:

- `+=` → ਜੋੜ ਕੇ ਅਸਾਈਨ ਕਰਨਾ: `a += 5;` ਇਹ `a = a + 5;` ਦੇ ਬਰਾਬਰ ਹੈ
- `-=` → ਘਟਾ ਕੇ ਅਸਾਈਨ ਕਰਨਾ: `a -= 3;`
- `*=` → ਗੁਣਾ ਕਰਕੇ ਅਸਾਈਨ ਕਰਨਾ: `a *= 2;`
- `/=` → ਭਾਗ ਦੇ ਕੇ ਅਸਾਈਨ ਕਰਨਾ: `a /= 4;`
- `%=` → Modulus ਕਰਕੇ ਅਸਾਈਨ ਕਰਨਾ: `a %= 2;`
- `&=`, `|=`, `^=`, `<<=`, `>>=` → ਇਹ Bitwise assignment operators ਹਨ

ਇਹ operators ਕੋਡ ਵਿੱਚ ਦੁਹਰਾਅ ਨੂੰ ਘਟਾਉਂਦੇ ਹਨ ਅਤੇ ਸਮਝਣ ਵਿੱਚ ਆਸਾਨ ਹੁੰਦੇ ਹਨ।

Pre ਅਤੇ Post Increment/Decrement ਵਿੱਚ ਅੰਤਰ:

Pre-Increment/Decrement (`++i`, `--i`): ਵੈਰੀਏਬਲ ਦੀ value ਨੂੰ ਪਹਿਲਾਂ ਵਧਾਇਆ ਜਾਂ ਘਟਾਇਆ ਜਾਂਦਾ ਹੈ, ਫਿਰ ਉਹ value expression ਵਿੱਚ ਵਰਤੀ ਜਾਂਦੀ ਹੈ।

ਉਦਾਹਰਨ:

```
int a = 5, b;
```

```
b = ++a; // ਪਹਿਲਾਂ a = 6, ਫਿਰ b = 6
```

Post-Increment/Decrement (i++, i--): ਵੈਰੀਏਬਲ ਦੀ ਮੂਲ value ਨੂੰ ਪਹਿਲਾਂ expression ਵਿੱਚ ਵਰਤਿਆ ਜਾਂਦਾ ਹੈ, ਫਿਰ value ਵਿੱਚ ਤਬਦੀਲੀ ਕੀਤੀ ਜਾਂਦੀ ਹੈ।

ਉਦਾਹਰਨ:

```
int a = 5, b;
```

```
b = a++; // ਪਹਿਲਾਂ b = 5, ਫਿਰ a = 6
```

Pre ਅਤੇ Post operations ਆਮ ਤੌਰ 'ਤੇ loops ਅਤੇ expressions ਵਿੱਚ ਵਰਤੇ ਜਾਂਦੇ ਹਨ ਜਿੱਥੇ value ਨੂੰ ਨਿਯੰਤ੍ਰਿਤ ਢੰਗ ਨਾਲ ਅੱਪਡੇਟ ਕਰਨਾ ਲੋੜੀਂਦਾ ਹੈ।

Q. Write a C++ program to swap two numbers with and without the use of third variable. (Nov 20)

Ans.

```
#include <iostream>
```

```
using namespace std;
```

```
int main() {
```

```
    int a, b, temp;
```

```
    // Input two numbers
```

```
    cout << "Enter two numbers: ";
```

```
    cin >> a >> b;
```

```
    // Swap using third variable
```

```
    cout << "\nSwapping with third variable..." << endl;
```

```
    temp = a;
```

```
    a = b;
```

```
    b = temp;
```

```
    cout << "After swap: a = " << a << ", b = " << b << endl;
```

```
    // Reset values
```

```
    cout << "\nEnter two new numbers: ";
```

```
    cin >> a >> b;
```

```
    // Swap without using third variable
```

```
    cout << "\nSwapping without third variable..." << endl;
```

```
    a = a + b;
```

```
    b = a - b;
```

```
    a = a - b;
```

```
    cout << "After swap: a = " << a << ", b = " << b << endl;
```

```
    return 0;
```

```
}
```

This program demonstrates two methods of swapping variables:

1. **With a third variable:** Uses a temporary variable temp to hold the value of one variable during swapping.
2. **Without a third variable:** Uses arithmetic operations (addition and subtraction) to swap the values.

Both methods achieve the same result but are useful in different situations. The second method is memory efficient but must be used carefully to avoid overflow.

1. ਤੀਜੇ ਵੈਰੀਏਬਲ ਨਾਲ:

ਇਸ ਤਰੀਕੇ ਵਿੱਚ ਇੱਕ ਅਸਥਾਈ ਵੈਰੀਏਬਲ temp ਵਰਤਿਆ ਜਾਂਦਾ ਹੈ ਜੋ ਇੱਕ ਵੈਰੀਏਬਲ ਦੀ value ਨੂੰ temporarily ਸਾਂਭ ਕੇ ਰੱਖਦਾ ਹੈ ਜਦ ਤੱਕ ਦੂਜੇ ਦੀ value ਅਸਾਈਨ ਕੀਤੀ ਜਾਂਦੀ ਹੈ।

2. ਤੀਜੇ ਵੈਰੀਏਬਲ ਤੋਂ ਬਿਨਾਂ:

ਇਸ ਤਰੀਕੇ ਵਿੱਚ ਗਣਿਤਿਕ ਕ੍ਰਿਯਾਵਾਂ (ਜੋੜ ਅਤੇ ਘਟਾਓ) ਦੀ ਵਰਤੋਂ ਕਰਕੇ values ਦੀ ਅਦਲਾ-ਬਦਲੀ ਕੀਤੀ ਜਾਂਦੀ ਹੈ।

ਦੇਵਾਂ ਤਰੀਕੇ ਇੱਕੋ ਹੀ ਨਤੀਜਾ ਦਿੰਦੇ ਹਨ ਪਰ ਵੱਖ-ਵੱਖ ਹਾਲਤਾਂ ਵਿੱਚ ਲਾਭਕਾਰੀ ਹੁੰਦੇ ਹਨ। ਦੂਜਾ ਤਰੀਕਾ memory efficient ਹੁੰਦਾ ਹੈ ਪਰ ਇਸ ਦੀ ਵਰਤੋਂ ਸਮੇਂ overflow ਤੋਂ ਬਚਣ ਲਈ ਧਿਆਨ ਦੀ ਲੋੜ ਹੁੰਦੀ ਹੈ।

Q. Write a program to check whether a number is palindrome or not? (Nov 20)

Ans. #include <iostream>

using namespace std;

```
int main() {
    int num, reversed = 0, remainder, original;

    // Input number from user
    cout << "Enter an integer: ";
    cin >> num;

    original = num; // Store original number for comparison

    // Reverse the number
    while (num != 0) {
        remainder = num % 10; // Get last digit
        reversed = reversed * 10 + remainder; // Build reversed number
        num /= 10; // Remove last digit
    }

    // Check if original and reversed are the same
    if (original == reversed)
        cout << original << " is a palindrome." << endl;
    else
        cout << original << " is not a palindrome." << endl;

    return 0;
}
```

A **palindrome** number is one that remains the same when its digits are reversed (e.g., 121, 1331).

This program:

- Takes an integer input from the user.
- Stores the original number.
- Reverses the number using a while loop.
- Compares the reversed number with the original.
- If both are the same, it prints that the number is a palindrome; otherwise, it is not.

This program demonstrates use of loops, conditionals, and arithmetic operations in C++.

Palindrome ਨੰਬਰ ਉਹ ਨੰਬਰ ਹੁੰਦੇ ਹਨ ਜੋ ਉਲਟੇ ਕਰਨ 'ਤੇ ਵੀ ਉਹੀ ਰਹਿੰਦੇ ਹਨ (ਜਿਵੇਂ 121, 1331)।

ਇਹ ਕਾਰਜ (program) ਇਹ ਦਰਸਾਉਂਦਾ ਹੈ:

- ਯੂਜ਼ਰ ਤੋਂ ਇੱਕ integer ਇਨਪੁੱਟ ਲੈਂਦਾ ਹੈ।
- Original ਨੰਬਰ ਨੂੰ ਸੰਭਾਲ ਕੇ ਰੱਖਦਾ ਹੈ।
- while ਲੂਪ ਦੀ ਮਦਦ ਨਾਲ ਨੰਬਰ ਨੂੰ ਉਲਟਦਾ ਹੈ।
- Reversed ਨੰਬਰ ਨੂੰ Original ਨੰਬਰ ਨਾਲ ਤੁਲਨਾ ਕਰਦਾ ਹੈ।

- ਜੇ ਦੋਵੇਂ ਨੰਬਰ ਇਕੱਥੇ ਹਨ ਤਾਂ ਪ੍ਰਿੰਟ ਕਰਦਾ ਹੈ ਕਿ ਇਹ ਨੰਬਰ palindrome ਹੈ, ਨਹੀਂ ਤਾਂ ਦੱਸਦਾ ਹੈ ਕਿ ਇਹ palindrome ਨਹੀਂ ਹੈ।

ਇਹ ਕਾਰਜ loops, conditionals ਅਤੇ arithmetic operations ਦੀ ਵਰਤੋਂ ਨੂੰ C++ ਵਿੱਚ ਵਿਖਾਉਂਦਾ ਹੈ।

Q. What is the meaning of scope of a variable in C++? Illustrate with an example. Write in detail about Pass by Value and Pass by Reference. (Nov 22)

Ans. Scope of a variable refers to the part of a program where the variable is accessible or can be used. In C++, scopes are categorized as:

1. **Local Scope:** Variables declared inside a function or block and accessible only within that block.
2. **Global Scope:** Variables declared outside all functions and accessible throughout the program.
3. **Block Scope:** Variables declared within control statements like if, for, or {} blocks.

Example:

```
#include <iostream>
using namespace std;
int x = 10; // Global scope

void show() {
    int x = 5; // Local scope
    cout << "Local x: " << x << endl;
}

int main() {
    show();
    cout << "Global x: " << x << endl;
    return 0;
}
```

Pass by Value vs. Pass by Reference:

- **Pass by Value:** A copy of the variable is passed to the function. Changes made inside the function do not affect the original variable.

```
void func(int x) { x = x + 5; }
```

- **Pass by Reference:** The actual variable is passed using reference (&). Changes inside the function affect the original variable.

```
void func(int &x) { x = x + 5; }
```

Use **pass by reference** when you want to modify original values or improve performance by avoiding copies.

ਵੈਰੀਅਬਲ ਦੀ Scope: ਵੈਰੀਅਬਲ ਦੀ scope ਤੋਂ ਭਾਵ ਹੈ ਕਿ ਕਿਸ ਹਿੱਸੇ ਵਿੱਚ ਉਸ ਵੈਰੀਅਬਲ ਨੂੰ ਐਕਸੈੱਸ ਕੀਤਾ ਜਾ ਸਕਦਾ ਹੈ ਜਾਂ ਵਰਤਿਆ ਜਾ ਸਕਦਾ ਹੈ। C++ ਵਿੱਚ ਇਹ ਤਿੰਨ ਕਿਸਮਾਂ ਦੀ ਹੋ ਸਕਦੀ ਹੈ:

1. **Local Scope (ਸਥਾਨਕ ਸੀਮਾ):** ਵੈਰੀਅਬਲ ਜੋ ਕਿਸੇ function ਜਾਂ block ਵਿੱਚ ਡਿਕਲੇਅਰ ਕੀਤਾ ਜਾਂਦਾ ਹੈ ਅਤੇ ਸਿਰਫ਼ ਉਨ੍ਹਾਂ ਦੇ ਅੰਦਰ ਹੀ ਵਰਤਿਆ ਜਾ ਸਕਦਾ ਹੈ।
2. **Global Scope (ਗਲੋਬਲ ਸੀਮਾ):** ਵੈਰੀਅਬਲ ਜੋ ਸਾਰੇ functions ਤੋਂ ਬਾਹਰ ਡਿਕਲੇਅਰ ਕੀਤਾ ਜਾਂਦਾ ਹੈ ਅਤੇ ਪੂਰੇ ਪ੍ਰੋਗਰਾਮ ਵਿੱਚ ਉਪਲਬਧ ਹੁੰਦਾ ਹੈ।
3. **Block Scope (ਬਲਾਕ ਸੀਮਾ):** ਵੈਰੀਅਬਲ ਜੋ if, for, ਜਾਂ {} ਵਰਗੇ statements ਵਿੱਚ ਡਿਕਲੇਅਰ ਹੁੰਦੇ ਹਨ ਅਤੇ ਉਨ੍ਹਾਂ blocks ਵਿੱਚ ਹੀ ਵਰਤਿਆ ਜਾ ਸਕਦਾ ਹੈ।

ਉਦਾਹਰਨ:

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

```

int x = 10; // Global scope

void show() {
    int x = 5; // Local scope
    cout << "Local x: " << x << endl;
}

int main() {
    show();
    cout << "Global x: " << x << endl;
    return 0;
}

```

Pass by Value vs Pass by Reference

Pass by Value (ਮੂਲ ਰਾਹੀਂ ਪਾਸ ਕਰਨਾ): ਇਸ ਵਿੱਚ function ਨੂੰ ਵੈਰੀਅਬਲ ਦੀ copy ਦਿੱਤੀ ਜਾਂਦੀ ਹੈ। Function ਵਿੱਚ ਹੋਈਆਂ ਤਬਦੀਲੀਆਂ ਮੂਲ ਵੈਰੀਅਬਲ ਨੂੰ ਪ੍ਰਭਾਵਿਤ ਨਹੀਂ ਕਰਦੀਆਂ।

```
void func(int x) { x = x + 5; }
```

Pass by Reference (ਹਵਾਲਾ ਰਾਹੀਂ ਪਾਸ ਕਰਨਾ): ਇਸ ਵਿੱਚ function ਨੂੰ ਵੈਰੀਅਬਲ ਦਾ ਹਵਾਲਾ (&) ਦਿੱਤਾ ਜਾਂਦਾ ਹੈ। Function ਵਿੱਚ ਕੀਤੀ ਤਬਦੀਲੀ ਮੂਲ ਵੈਰੀਅਬਲ ਵਿੱਚ ਹੋ ਜਾਂਦੀ ਹੈ।

```
void func(int &x) { x = x + 5; }
```

ਕਿਉਂ ਵਰਤਦੇ ਹਾਂ Pass by Reference?

- ਜਦੋਂ ਤੁਸੀਂ ਮੂਲ ਡੇਟਾ ਨੂੰ ਤਬਦੀਲ ਕਰਨਾ ਚਾਹੁੰਦੇ ਹੋ।
- ਜਦੋਂ ਤੁਸੀਂ copy ਬਣਾਉਣ ਤੋਂ ਬਚਣਾ ਚਾਹੁੰਦੇ ਹੋ (ਪ੍ਰਦਰਸ਼ਨ ਵਧਾਉਣ ਲਈ)।

Unit 2: Control Statements and Functions

Short Answer Questions:

Q. What do you mean by if-else ladder? (Nov 22)

Ans. An if-else ladder is used to evaluate multiple conditions sequentially, executing the block of the first true condition.

if-else ladder ਕਈ ਸ਼ਰਤਾਂ ਨੂੰ ਲਗਾਤਾਰ ਜਾਂਚਣ ਲਈ ਵਰਤੀ ਜਾਂਦੀ ਹੈ ਅਤੇ ਜਿਸ ਵੀ ਸ਼ਰਤ ਨੂੰ ਸਭ ਤੋਂ ਪਹਿਲਾਂ ਸੱਚ ਪਾਇਆ ਜਾਂਦਾ ਹੈ, ਉਸਦਾ ਕੋਡ ਬਲਾਕ ਚਲਾਇਆ ਜਾਂਦਾ ਹੈ।

Q. What is the advantage of switch statement over If-Else statement? (Nov 20)

Ans. The switch statement is more readable and efficient when handling multiple discrete values of a single variable.

switch statement ਇੱਕ ਹੀ ਵੇਰੀਏਬਲ ਦੀ ਕਈ ਅਲੱਗ-ਅਲੱਗ ਮੁੱਲਾਂ ਨੂੰ ਸੰਭਾਲਣ ਵੇਲੇ ਜ਼ਿਆਦਾ ਪੜ੍ਹਨਯੋਗ ਅਤੇ ਪ੍ਰਭਾਵਸ਼ਾਲੀ ਹੁੰਦੀ ਹੈ।

Q. Global Variable (Nov 24)

Ans. A global variable is declared outside all functions and is accessible throughout the entire program.

Global variable ਉਹ ਵੇਰੀਏਬਲ ਹੁੰਦੀ ਹੈ ਜੋ ਸਾਰੀਆਂ ਫੰਕਸ਼ਨਾਂ ਤੋਂ ਬਾਹਰ ਘੋਸ਼ਿਤ ਕੀਤੀ ਜਾਂਦੀ ਹੈ ਅਤੇ ਪੂਰੇ ਪ੍ਰੋਗ੍ਰਾਮ ਵਿੱਚ ਉਪਲਬਧ ਰਹਿੰਦੀ ਹੈ।

Q. Static variable (Nov 24)

Ans. A static variable retains its value between function calls and is initialized only once.

Static variable ਇੱਕ ਐਸੀ ਵੇਰੀਏਬਲ ਹੁੰਦੀ ਹੈ ਜੋ ਫੰਕਸ਼ਨ ਕਾਲਾਂ ਵਿਚਕਾਰ ਆਪਣੀ ਕੀਮਤ ਸੰਭਾਲ ਕੇ ਰੱਖਦੀ ਹੈ ਅਤੇ ਸਿਰਫ ਇੱਕ ਵਾਰੀ ਹੀ ਇਨਿਸ਼ੀਅਲਾਈਜ਼ ਹੁੰਦੀ ਹੈ।

Q. Protected (Nov 24)

Ans. protected is an access specifier that allows a class member to be accessed by the class itself and its derived classes.

protected ਇੱਕ access specifier ਹੁੰਦਾ ਹੈ ਜੋ ਕਿਸੇ ਕਲਾਸ ਮੈਂਬਰ ਨੂੰ ਉਸ ਕਲਾਸ ਅਤੇ ਉਸ ਤੋਂ ਨਿਕਲੀ ਹੋਈ ਡਿਰਾਈਵਡ ਕਲਾਸਾਂ ਵੱਲੋਂ ਐਕਸੈੱਸ ਕਰਨ ਦੀ ਆਗਿਆ ਦਿੰਦਾ ਹੈ।

Q. Write a program to find the mean of 5 numbers. (Nov 20)

```
Ans. #include <iostream>
using namespace std;
int main() {
    float a, b, c, d, e, mean;
    cin >> a >> b >> c >> d >> e;
    mean = (a + b + c + d + e) / 5;
    cout << "Mean = " << mean;
    return 0;
}
```

Long Answer Questions:

Q. Explain the use of functions and its types. (Nov 24)

Ans. **Functions** in C++ are blocks of code designed to perform specific tasks. They help in modularizing the program, promoting code reusability, and making it easier to test, debug, and maintain.

Uses of Functions:

- **Code Reusability:** A function can be called multiple times in a program.

- **Modularity:** Breaks down a complex program into manageable sections.
- **Improved Readability:** Each function performs a specific task, making the program easier to understand.
- **Ease of Maintenance:** Modifying one function doesn't affect others.

Types of Functions:

1. **Library Functions:** These are built-in functions provided by C++ libraries, such as `sqrt()`, `pow()`, `cin`, and `cout`.
2. **User-defined Functions:** These are functions created by the programmer to perform specific tasks.

With no return & no parameters:

```
void display() { cout << "Hello"; }
```

With return & no parameters:

```
int getValue() { return 10; }
```

With return & parameters:

```
int sum(int a, int b) { return a + b; }
```

Functions enhance program structure and are essential for large-scale development. Proper use of function types ensures efficient and clean coding practices.

C++ ਵਿੱਚ Functions: Functions ਉਹ ਕੋਡ ਦੇ block ਹੁੰਦੇ ਹਨ ਜੋ ਕਿਸੇ ਨਿਰਧਾਰਤ ਕੰਮ ਨੂੰ ਕਰਨ ਲਈ ਬਣਾਏ ਜਾਂਦੇ ਹਨ। ਇਹ ਪ੍ਰੋਗਰਾਮ ਨੂੰ ਮੋਡੀਊਲ ਬਣਾਉਂਦੇ ਹਨ, ਕੋਡ ਦੀ ਦੁਬਾਰਾ ਵਰਤੋਂ ਕਰਵਾਉਂਦੇ ਹਨ, ਅਤੇ ਟੈਸਟ, ਡੀਬੱਗ ਅਤੇ ਮੈਂਟੇਨ ਕਰਨਾ ਆਸਾਨ ਬਣਾਉਂਦੇ ਹਨ।

Functions ਦੇ ਉਪਯੋਗ (Uses of Functions):

- **Code Reusability (ਕੋਡ ਦੀ ਦੁਬਾਰਾ ਵਰਤੋਂ):** ਇੱਕ Function ਨੂੰ ਕਈ ਵਾਰੀ ਕਾਲ ਕੀਤਾ ਜਾ ਸਕਦਾ ਹੈ।
- **Modularity (ਮੋਡੀਊਲ ਬਣਾਉਣਾ):** ਵੱਡੇ ਪ੍ਰੋਗਰਾਮ ਨੂੰ ਛੋਟੇ-ਛੋਟੇ ਹਿੱਸਿਆਂ ਵਿੱਚ ਵੰਡਣਾ।
- **Readability ਵਧਾਉਂਦੀ ਹੈ:** ਹਰ Function ਇੱਕ ਖਾਸ ਕੰਮ ਕਰਦਾ ਹੈ ਜਿਸ ਨਾਲ ਪ੍ਰੋਗਰਾਮ ਸਮਝਣ ਵਿੱਚ ਆਸਾਨੀ ਹੁੰਦੀ ਹੈ।
- **Maintenance ਆਸਾਨ ਬਣਦੀ ਹੈ:** ਇੱਕ Function ਵਿੱਚ ਤਬਦੀਲੀ ਕਰਨ ਨਾਲ ਹੋਰ Functions ਪ੍ਰਭਾਵਿਤ ਨਹੀਂ ਹੁੰਦੇ।

Functions ਦੀਆਂ ਕਿਸਮਾਂ (Types of Functions):

1. Library Functions (ਲਾਇਬ੍ਰੇਰੀ ਫੰਕਸ਼ਨ):

C++ ਵਿੱਚ ਪੂਰੀ ਤਰ੍ਹਾਂ ਬਣੇ ਹੋਏ Functions, ਜਿਵੇਂ:

- `sqrt()` – ਵਰਗਮੂਲ
- `pow()` – ਘਾਤ
- `cin`, `cout` – ਇਨਪੁਟ/ਆਉਟਪੁਟ

2. User-defined Functions (ਵਰਤੋਂਕਾਰ ਵੱਲੋਂ ਬਣਾਏ ਗਏ):

ਯੂਜ਼ਰ ਦੁਆਰਾ ਨਿਰਧਾਰਤ ਕੰਮਾਂ ਲਈ ਬਣਾਏ ਜਾਂਦੇ Functions।

Examples (ਉਦਾਹਰਨਾਂ):

◆ Return ਨਾ ਹੋਵੇ, ਨਾ ਹੀ Parameters:

```
void display() {
    cout << "Hello";
}
```

◆ Return ਹੋਵੇ, ਪਰ Parameters ਨਾ ਹੋਣ:

```
int getValue() {  
    return 10;  
}
```

◆ Return ਵੀ ਹੋਵੇ ਅਤੇ Parameters ਵੀ ਹੋਣ:

```
int sum(int a, int b) {  
    return a + b;  
}
```

ਸੰਖੇਪ ਵਿੱਚ: Functions ਪ੍ਰੋਗਰਾਮ ਦੇ ਢਾਂਚੇ ਨੂੰ ਸੁਧਾਰਦੇ ਹਨ ਅਤੇ ਵੱਡੇ ਪ੍ਰੋਜੈਕਟ ਲਈ ਬਹੁਤ ਲਾਭਕਾਰੀ ਹੁੰਦੇ ਹਨ। Functions ਦੀ ਸਹੀ ਵਰਤੋਂ ਸਾਫ਼, ਸਮਝਦਾਰ ਅਤੇ ਦੋਹਰਾਏ ਜਾ ਸਕਣ ਵਾਲਾ ਕੋਡ ਲਿਖਣ ਵਿੱਚ ਮਦਦ ਕਰਦੀ ਹੈ।

Q. Write short note on parameter passing in functions. (Nov 20)

Ans. Parameter passing in C++ refers to how arguments are sent to functions when they are called. It allows functions to accept input values and optionally modify them. There are two primary methods of parameter passing in C++:

1. Pass by Value:

- In this method, a copy of the actual argument is passed to the function.
- Any changes made to the parameter inside the function do **not affect** the original variable.
- It is safe, as original data is protected.

Example:

```
void update(int x) {  
    x = x + 5; // only local x is changed  
}
```

2. Pass by Reference:

- Instead of copying, the actual memory address of the argument is passed using reference (&).
- Changes made in the function **do affect** the original variable.
- Useful when the function needs to modify input or avoid unnecessary copying (especially for large objects).

Example:

```
void update(int &x) {  
    x = x + 5; // original x is changed  
}
```

Conclusion: Choosing between pass by value and reference depends on whether the original data needs to be modified. C++ also supports pointers as an alternative way to pass by reference, offering more control in memory management.

C++ ਵਿੱਚ Parameter Passing (ਪੈਰਾਮੀਟਰ ਪਾਸਿੰਗ): C++ ਵਿੱਚ Parameter Passing ਦਾ ਅਰਥ ਹੈ ਕਿ ਕਿਵੇਂ arguments (ਦਿੱਤੇ ਗਏ ਮੁੱਲ) ਇੱਕ Function ਨੂੰ ਕਾਲ ਕਰਦੇ ਸਮੇਂ ਭੇਜੇ ਜਾਂਦੇ ਹਨ। ਇਹ Functions ਨੂੰ ਇਨਪੁਟ ਲੈਣ ਅਤੇ ਜ਼ਰੂਰਤ ਪੈਣ ਤੇ ਮੁੱਲਾਂ ਨੂੰ ਤਬਦੀਲ ਕਰਨ ਦੀ ਸਹੂਲਤ ਦਿੰਦਾ ਹੈ।

C++ ਵਿੱਚ ਦੋ ਮੁੱਖ ਤਰੀਕੇ ਹਨ:

1. Pass by Value (ਮੁੱਲ ਰਾਹੀਂ ਭੇਜਣਾ):

- ਇਸ ਤਰੀਕੇ ਵਿੱਚ, argument ਦੀ ਇੱਕ ਕਾਪੀ Function ਨੂੰ ਭੇਜੀ ਜਾਂਦੀ ਹੈ।
- Function ਦੇ ਅੰਦਰ ਕੀਤੇ ਜਾਣ ਵਾਲੇ ਬਦਲਾਅ ਮੂਲ variable 'ਤੇ ਪ੍ਰਭਾਵ ਨਹੀਂ ਪਾਉਂਦੇ।
- ਇਹ ਸੁਰੱਖਿਅਤ ਹੁੰਦਾ ਹੈ ਕਿਉਂਕਿ ਅਸਲੀ ਡਾਟਾ ਸੁਰੱਖਿਅਤ ਰਹਿੰਦਾ ਹੈ।

◆ ਉਦਾਹਰਨ:

```
void update(int x) {  
    x = x + 5; // ਸਿਰਫ x ਦੀ ਕਾਪੀ ਵਿੱਚ ਬਦਲਾਅ  
}
```

2. Pass by Reference (ਹਵਾਲੇ ਰਾਹੀਂ ਭੇਜਣਾ):

- ਇਸ ਤਰੀਕੇ ਵਿੱਚ, argument ਦੀ ਮੈਮੋਰੀ location (address) Function ਨੂੰ ਭੇਜੀ ਜਾਂਦੀ ਹੈ, & ਦੀ ਵਰਤੋਂ ਕਰਕੇ।
- Function ਦੇ ਅੰਦਰ ਕੀਤੇ ਜਾਣ ਵਾਲੇ ਬਦਲਾਅ ਅਸਲੀ variable ਨੂੰ ਪ੍ਰਭਾਵਿਤ ਕਰਦੇ ਹਨ।
- ਇਹ ਤਦੋਂ ਲਾਭਕਾਰੀ ਹੁੰਦਾ ਹੈ ਜਦੋਂ:
 - Function ਨੇ input ਨੂੰ ਤਬਦੀਲ ਕਰਨਾ ਹੋਵੇ।
 - ਜਾਂ ਵੱਡੀ Object ਦੀ ਕਾਪੀ ਬਣਾਉਣ ਤੋਂ ਬਚਣਾ ਹੋਵੇ।

◆ ਉਦਾਹਰਨ:

```
void update(int &x) {  
    x = x + 5; // ਅਸਲੀ x now changed  
}
```

ਨਤੀਜਾ (Conclusion):

- ਜੇ ਅਸਲੀ ਡਾਟਾ ਨੂੰ ਤਬਦੀਲ ਨਹੀਂ ਕਰਨਾ, ਤਾਂ Pass by Value ਵਰਤੋਂ।
- ਜੇ ਤਬਦੀਲੀ ਕਰਨੀ ਹੋਵੇ ਜਾਂ efficient code ਚਾਹੀਦਾ ਹੋਵੇ, ਤਾਂ Pass by Reference ਵਰਤੋਂ।

💡 C++ ਵਿੱਚ pointers ਦੀ ਵਰਤੋਂ ਕਰਕੇ ਵੀ reference ਦੇ ਤਰੀਕੇ ਨਾਲ values pass ਕੀਤੀਆਂ ਜਾ ਸਕਦੀਆਂ ਹਨ, ਜੋ memory management 'ਤੇ ਹੋਰ control ਦਿੰਦੇ ਹਨ।

Q. What is the difference between a while and a do while loop? Explain with examples. (Nov 22)
Ans. In C++, both while and do-while loops are used for **repeating a block of code** as long as a specified condition is true. However, they differ in how and **when** the condition is evaluated.

while Loop:

- The condition is checked **before** the loop body executes.
- If the condition is false initially, the loop body may **never execute**.

Syntax:

```
int i = 1;  
while (i <= 3) {  
    cout << i << " ";  
    i++;  
}
```

Output: 1 2 3

do-while Loop:

- The loop body is executed **at least once**, regardless of the condition.
- The condition is checked **after** executing the loop body.

Syntax:

```
int i = 1;  
do {  
    cout << i << " ";  
    i++;  
} while (i <= 3);
```

Output: 1 2 3

Key Difference:

- **while loop:** Entry-controlled (condition checked first).
- **do-while loop:** Exit-controlled (condition checked after execution).

Use do-while when the loop must run at least once, such as in menu-driven programs or user input validation.

C++ ਵਿੱਚ while ਅਤੇ do-while ਲੂਪ

C++ ਵਿੱਚ while ਅਤੇ do-while ਲੂਪ ਦੋਵੇਂ ਇੱਕ ਕੋਡ ਦੇ block ਨੂੰ ਦੁਹਰਾਉਣ ਲਈ ਵਰਤੇ ਜਾਂਦੇ ਹਨ ਜਦ ਤੱਕ ਕੋਈ ਦਿੱਤੀ ਗਈ condition true ਰਹਿੰਦੀ ਹੈ। ਪਰ ਦੋਹਾਂ ਵਿੱਚ condition check ਕਰਨ ਦਾ ਤਰੀਕਾ ਵੱਖਰਾ ਹੁੰਦਾ ਹੈ।

✓ while Loop:

- Condition ਪਹਿਲਾਂ check ਕੀਤੀ ਜਾਂਦੀ ਹੈ।
- ਜੇ condition ਸ਼ੁਰੂ ਤੋਂ hi false ਹੋਵੇ, ਤਾਂ loop ਦੇ ਅੰਦਰਲਾ code ਇਕ ਵਾਰੀ ਵੀ execute ਨਹੀਂ ਹੁੰਦਾ।

✚ Syntax:

```
int i = 1;
while (i <= 3) {
    cout << i << " ";
    i++;
}
```

🖨 Output:

1 2 3

✓ do-while Loop:

- Loop body ਪਹਿਲਾਂ ਇੱਕ ਵਾਰੀ execute ਹੁੰਦੀ ਹੈ, ਭਾਵੇਂ condition false ਹੋਵੇ।
- Condition ਫਾਦ ਵਿੱਚ check ਹੁੰਦੀ ਹੈ।

✚ Syntax:

```
int i = 1;
do {
    cout << i << " ";
    i++;
} while (i <= 3);
```

🖨 Output:

1 2 3

🔄 ਮੁੱਖ ਅੰਤਰ (Key Difference):

Feature	while Loop	do-while Loop
Condition Check	ਪਹਿਲਾਂ (Entry-Controlled)	ਬਾਅਦ ਵਿੱਚ (Exit-Controlled)
Execution Guarantee	0 ਜਾਂ ਵੱਧ ਵਾਰ	ਘੱਟੋ-ਘੱਟ 1 ਵਾਰ
ਉਦੇਸ਼ (Use Case)	ਜਦ ਸ਼ਰਤ ਪਹਿਲਾਂ ਹੀ ਪਤਾ ਹੋਵੇ	ਜਦੋਂ code ਇੱਕ ਵਾਰੀ ਚਲਾਉਣਾ ਜ਼ਰੂਰੀ ਹੋਵੇ (ਜਿਵੇਂ ਕਿ menu-driven programs)

💡 **ਉਪਯੋਗਤਾ (When to Use):** 🖱️ do-while loop ਨੂੰ ਉਥੇ ਵਰਤਿਆ ਜਾਂਦਾ ਹੈ ਜਿੱਥੇ loop ਘੱਟੋ-ਘੱਟ ਇੱਕ ਵਾਰੀ ਚਲਾਉਣੀ ਲਾਜ਼ਮੀ ਹੋਵੇ, ਜਿਵੇਂ user input validation ਜਾਂ menus ਵਿੱਚ।

Q. Differentiate between the following: (Nov 20)

a) Formal and actual arguments

b) Call by value and Call by reference.

Ans. a) **Formal and Actual Arguments:**

Formal arguments are variables listed in a function's definition that receive values when the function is called. **Actual arguments** are the real values or variables passed to the function during the function call.

Example:

```
void add(int a, int b); // a, b are formal arguments
```

```
add(5, 10);           // 5, 10 are actual arguments
```

Formal arguments exist only within the function and are treated like local variables. Actual arguments are used in the calling function. The values of actual arguments are copied or referenced depending on the parameter passing method used.

b) **Call by Value and Call by Reference:**

In **call by value**, a copy of the actual argument is passed to the function. Changes made inside the function do **not** affect the original variable.

Example:

```
void modify(int x) { x = 10; }
```

In **call by reference**, the address (reference) of the actual argument is passed. Changes made in the function **do** affect the original variable.

Example:

```
void modify(int &x) { x = 10; }
```

Call by value is safe but doesn't allow modification of original data, while call by reference is more efficient and allows in-place changes.

a) **Formal ਅਤੇ Actual Arguments:** Formal arguments ਉਹ variables ਹੁੰਦੇ ਹਨ ਜੋ function ਦੀ definition ਵਿੱਚ ਲਿਖੇ ਜਾਂਦੇ ਹਨ ਅਤੇ function call ਸਮੇਂ ਇਹਨਾਂ ਨੂੰ value ਮਿਲਦੀ ਹੈ। Actual arguments ਉਹ ਅਸਲੀ values ਜਾਂ variables ਹੁੰਦੇ ਹਨ ਜੋ function ਨੂੰ call ਕਰਦੇ ਸਮੇਂ ਦਿੱਤੇ ਜਾਂਦੇ ਹਨ।

ਉਦਾਹਰਨ:

```
void add(int a, int b); // a, b- formal arguments
```

```
add(5, 10);           // 5, 10- actual arguments
```

Formal arguments function ਦੇ ਅੰਦਰ ਹੀ ਰਹਿੰਦੇ ਹਨ ਅਤੇ local variables ਵਾਂਗ ਕੰਮ ਕਰਦੇ ਹਨ। Actual arguments calling function ਵਿੱਚ ਵਰਤੇ ਜਾਂਦੇ ਹਨ। Values pass ਕਰਨ ਦਾ ਤਰੀਕਾ (copy ਜਾਂ reference) parameter passing technique 'ਤੇ ਨਿਰਭਰ ਕਰਦਾ ਹੈ।

b) **Call by Value ਅਤੇ Call by Reference:** Call by value ਵਿੱਚ actual argument ਦੀ copy function ਨੂੰ ਦਿੱਤੀ ਜਾਂਦੀ ਹੈ। Function ਵਿੱਚ ਕੀਤੇ ਗਏ ਤਬਦੀਲੀਆਂ original variable 'ਤੇ ਅਸਰ ਨਹੀਂ ਕਰਦੀਆਂ।

ਉਦਾਹਰਨ:

```
void modify(int x) { x = 10; }
```

Call by reference ਵਿੱਚ variable ਦਾ address (reference) function ਨੂੰ ਦਿੱਤਾ ਜਾਂਦਾ ਹੈ। Function ਵਿੱਚ ਕੀਤੇ ਤਬਦੀਲੀਆਂ actual variable ਨੂੰ ਪ੍ਰਭਾਵਿਤ ਕਰਦੀਆਂ ਹਨ।

ਉਦਾਹਰਨ:

```
void modify(int &x) { x = 10; }
```

Call by value ਸੁਰੱਖਿਤ ਹੈ ਪਰ original value ਨੂੰ change ਨਹੀਂ ਕਰਦਾ। Call by reference efficient ਹੈ ਅਤੇ directly value ਵਿੱਚ change ਕਰਦਾ ਹੈ।

Q. Write difference between call by value, call by address and call by reference explain with the help examples. (Nov 24)

Ans. In C++, **function arguments** can be passed in three main ways: **call by value**, **call by address**, and **call by reference**. These differ in how the function accesses and modifies data.

1. Call by Value:

- A **copy** of the variable is passed to the function.
- Changes inside the function **do not affect** the original variable.

Example:

```
void modify(int x) { x = 10; }
```

Calling modify(a) does **not** change the value of a.

2. Call by Address:

- The **memory address** of the variable is passed using pointers.
- The function can **modify the original value** by dereferencing the pointer.

Example:

```
void modify(int *x) { *x = 10; }
```

Calling modify(&a) changes the value of a.

3. Call by Reference:

- A **reference** (alias) of the variable is passed using &.
- Changes inside the function **directly affect** the original variable.

Example:

```
void modify(int &x) { x = 10; }
```

Calling modify(a) updates the value of a.

Summary:

Type	Can Modify Original?	Syntax Used
Call by Value	No	func(int x)
Call by Address	Yes	func(int *x)
Call by Reference	Yes	func(int &x)

C++ ਵਿੱਚ Function Arguments ਪਾਸ ਕਰਨ ਦੇ ਤਿੰਨ ਤਰੀਕੇ:

C++ ਵਿੱਚ function arguments ਤਿੰਨ ਮੁੱਖ ਤਰੀਕਿਆਂ ਨਾਲ ਪਾਸ ਕੀਤੇ ਜਾਂਦੇ ਹਨ: Call by Value, Call by Address, ਅਤੇ Call by Reference। ਇਹ ਤਿੰਨੇ ਤਰੀਕੇ data ਨੂੰ access ਅਤੇ modify ਕਰਨ ਵਿੱਚ ਵੱਖ-ਵੱਖ ਤਰੀਕੇ ਨਾਲ ਕੰਮ ਕਰਦੇ ਹਨ।

1. Call by Value (ਮੂਲ ਦੇ ਮੁੱਲ ਰਾਹੀਂ):

- Variable ਦੀ ਇੱਕ copy function ਨੂੰ ਦਿੱਤੀ ਜਾਂਦੀ ਹੈ।
- Function ਵਿੱਚ ਹੋਣ ਵਾਲੀਆਂ ਤਬਦੀਲੀਆਂ original variable ਨੂੰ ਪ੍ਰਭਾਵਿਤ ਨਹੀਂ ਕਰਦੀਆਂ।

ਉਦਾਹਰਨ:

```
void modify(int x) { x = 10; }
```

modify(a) ਨੂੰ call ਕਰਨ 'ਤੇ a ਦੀ value change ਨਹੀਂ ਹੁੰਦੀ।

2. Call by Address (ਐਡਰੈੱਸ ਰਾਹੀਂ):

- Variable ਦਾ memory address function ਨੂੰ pointer ਰਾਹੀਂ ਦਿੱਤਾ ਜਾਂਦਾ ਹੈ।

- Function pointer ਨੂੰ dereference ਕਰਕੇ original value ਨੂੰ change ਕਰ ਸਕਦਾ ਹੈ।

ਉਦਾਹਰਨ:

```
void modify(int *x) { *x = 10; }
```

modify(&a) ਨੂੰ call ਕਰਨ 'ਤੇ a ਦੀ value change ਹੋ ਜਾਂਦੀ ਹੈ।

3. Call by Reference (ਹਵਾਲੇ ਰਾਹੀਂ):

- Variable ਦਾ reference (alias) function ਨੂੰ ਦਿੱਤਾ ਜਾਂਦਾ ਹੈ।
- Function ਵਿੱਚ ਹੋਈ ਤਬਦੀਲੀ seedhi original variable ਨੂੰ ਪ੍ਰਭਾਵਿਤ ਕਰਦੀ ਹੈ।

ਉਦਾਹਰਨ:

```
void modify(int &x) { x = 10; }
```

modify(a) ਨੂੰ call ਕਰਨ 'ਤੇ a ਦੀ value change ਹੋ ਜਾਂਦੀ ਹੈ।

ਸੰਖੇਪ ਤੌਰ 'ਤੇ:

ਤਰੀਕਾ	Original Value Change ਹੋਦੀ?	Syntax
Call by Value	ਨਹੀਂ	func(int x)
Call by Address	ਹਾਂ	func(int *x)
Call by Reference	ਹਾਂ	func(int &x)

Q. Write a program to print table of a number. (Nov 24)

Ans. #include <iostream>

using namespace std;

```
int main() {
    int num;

    // Input from user
    cout << "Enter a number to print its table: ";
    cin >> num;

    // Print table using loop
    cout << "\nMultiplication Table of " << num << ":\n";
    for (int i = 1; i <= 10; i++) {
        cout << num << " x " << i << " = " << num * i << endl;
    }

    return 0;
}
```

This program prints the **multiplication table** of any number provided by the user.

Steps involved:

1. The program begins by asking the user to enter an integer.
2. It uses a for loop that runs from 1 to 10.
3. In each iteration, the number is multiplied by the loop counter (i), and the result is printed in a standard format.

Sample Output for input 5:

5 x 1 = 5

5 x 2 = 10

...

$5 \times 10 = 50$

This program demonstrates use of cin for input, cout for output, and for loop for iteration — making it a basic yet essential C++ example for beginners.

ਇਹ Program ਕਿਸੇ ਵੀ ਨੰਬਰ ਦੀ ਗੁਣਾ ਪਾਂਤੀ (Multiplication Table) ਪ੍ਰਿੰਟ ਕਰਦਾ ਹੈ ਜੋ ਯੂਜ਼ਰ ਦੁਆਰਾ ਦਿੱਤਾ ਜਾਂਦਾ ਹੈ।

ਕੰਮ ਕਰਨ ਦੇ ਕਦਮ:

1. Program ਸਭ ਤੋਂ ਪਹਿਲਾਂ ਯੂਜ਼ਰ ਤੋਂ ਇੱਕ ਪੂਰਨ ਅੰਕ (integer) ਮੰਗਦਾ ਹੈ।
2. ਫਿਰ ਇੱਕ for loop ਚਲਦਾ ਹੈ ਜੋ 1 ਤੋਂ 10 ਤੱਕ ਜਾਂਦਾ ਹੈ।
3. ਹਰ iteration ਵਿੱਚ, ਉਹ ਨੰਬਰ loop ਦੇ counter (i) ਨਾਲ ਗੁਣਾ ਕੀਤਾ ਜਾਂਦਾ ਹੈ ਅਤੇ ਨਤੀਜਾ ਇੱਕ standard ਫਾਰਮੈਟ ਵਿੱਚ ਪ੍ਰਿੰਟ ਕੀਤਾ ਜਾਂਦਾ ਹੈ।

ਉਦਾਹਰਨ (Input = 5):

$5 \times 1 = 5$

$5 \times 2 = 10$

$5 \times 3 = 15$

...

$5 \times 10 = 50$

ਇਸ Program ਵਿੱਚ ਇਹ Concepts ਵਰਤੇ ਗਏ ਹਨ:

- cin → ਯੂਜ਼ਰ ਤੋਂ input ਲੈਣ ਲਈ
- cout → ਆਉਟਪੁੱਟ ਵਿਖਾਉਣ ਲਈ
- for loop → 1 ਤੋਂ 10 ਤੱਕ iterate ਕਰਨ ਲਈ

ਇਹ ਇੱਕ ਬਹੁਤ ਹੀ ਆਸਾਨ ਅਤੇ ਮੁਢਲੀ C++ ਦੀ ਉਦਾਹਰਨ ਹੈ ਜੋ Beginners ਲਈ ਬੇਹੱਦ ਲਾਭਕਾਰੀ ਹੈ।

Unit 3: Object Oriented Programming

Short Answer Questions:

Q. Why C++ is called object-oriented programming? (Nov 20)

Ans. C++ supports concepts like classes, objects, inheritance, and encapsulation, which makes it an object-oriented programming language.

C++ ਵਿੱਚ classes, objects, inheritance ਅਤੇ encapsulation ਵਰਗੀਆਂ ਵਿਸ਼ੇਸ਼ਤਾਵਾਂ ਹੁੰਦੀਆਂ ਹਨ, ਜੋ ਇਸਨੂੰ ਇੱਕ object-oriented programming language ਬਣਾਉਂਦੀਆਂ ਹਨ।

Q. What is operator overloading? (Nov 20)

Ans. Operator overloading allows redefining the meaning of operators for user-defined data types like classes.

Operator overloading ਦੇ ਜ਼ਰੀਏ ਅਸੀਂ user-defined data types (ਜਿਵੇਂ ਕਿ classes) ਲਈ ਓਪਰੇਟਰਾਂ ਦੀ ਪਰਿਭਾਸ਼ਾ ਨੂੰ ਦੁਬਾਰਾ ਨਿਰਧਾਰਤ ਕਰ ਸਕਦੇ ਹਾਂ।

Q. Destructor (Nov 24)

Ans. A destructor is a special member function that is automatically called when an object is destroyed to free resources.

Destructor ਇੱਕ ਖਾਸ member function ਹੁੰਦੀ ਹੈ ਜੋ ਕਿਸੇ object ਦੇ ਨਸ਼ਟ ਹੋਣ ਸਮੇਂ ਆਪਣੇ ਆਪ ਕਾਲ ਹੁੰਦੀ ਹੈ ਅਤੇ ਸਮੱਸਾਧਨਾਂ (resources) ਨੂੰ ਖਾਲੀ ਕਰਦੀ ਹੈ।

Q. Friend class (Nov 24)

Ans. A friend class can access the private and protected members of another class in which it is declared as a friend.

Friend class ਉਹ ਕਲਾਸ ਹੁੰਦੀ ਹੈ ਜੋ ਕਿਸੇ ਹੋਰ ਕਲਾਸ ਦੇ private ਅਤੇ protected ਮੈਂਬਰਾਂ ਨੂੰ ਐਕਸੈੱਸ ਕਰ ਸਕਦੀ ਹੈ, ਜਦੋਂ ਕਿ ਉਹ ਉਸ ਵਿੱਚ friend ਵਜੋਂ ਡਿਕਲੇਅਰ ਕੀਤੀ ਗਈ ਹੋਵੇ।

Q. Function overloading. (Nov 24)

ans. Function overloading allows multiple functions with the same name but different parameter lists within the same scope.

Function overloading ਇੱਕੋ ਨਾਮ ਵਾਲੀਆਂ ਕਈ functions ਨੂੰ ਇੱਕੋ scope ਵਿੱਚ ਪਰਿਭਾਸ਼ਤ ਕਰਨ ਦੀ ਆਗਿਆ ਦਿੰਦੀ ਹੈ, ਪਰ ਉਨ੍ਹਾਂ ਦੇ parameter lists ਵੱਖ-ਵੱਖ ਹੋਣੇ ਚਾਹੀਦੇ ਹਨ।

Q. When do we need iterator? (Nov 22)

Ans. Iterators are used to traverse containers like arrays, vectors, or lists in a standard and consistent way.

Iterators ਨੂੰ arrays, vectors ਜਾਂ lists ਵਰਗੇ containers ਵਿੱਚ ਰੇਖੀਤ ਢੰਗ ਨਾਲ ਟ੍ਰੈਵਰਸ ਕਰਨ ਲਈ ਵਰਤਿਆ ਜਾਂਦਾ ਹੈ।

Q. What is the usage of flush in C++ programming? (Nov 22)

Ans. flush is used to force the output buffer to write all data to the console or file immediately.

flush ਨੂੰ output buffer ਦੀ ਸਮੁੱਚੀ ਸਮੱਗਰੀ ਨੂੰ ਤੁਰੰਤ console ਜਾਂ file ਵਿੱਚ ਲਿਖਵਾਉਣ ਲਈ ਵਰਤਿਆ ਜਾਂਦਾ ਹੈ।

Q. Is destructor overloading possible? If yes then explain and if no then why? (Nov 22)

Ans. No, destructor overloading is not allowed in C++ because a class can have only one destructor with a fixed signature.

ਨਹੀਂ, C++ ਵਿੱਚ destructor overloading ਦੀ ਆਗਿਆ ਨਹੀਂ ਹੈ ਕਿਉਂਕਿ ਇੱਕ ਕਲਾਸ ਵਿੱਚ ਸਿਰਫ਼ ਇੱਕ ਹੀ destructor ਹੋ ਸਕਦੀ ਹੈ ਜਿਸ ਦੀ signature fixed ਹੁੰਦੀ ਹੈ।

Q. Define data abstraction. (Nov 22)

Ans. Data abstraction means exposing only essential features while hiding internal implementation details in a program.

Data abstraction ਦਾ ਅਰਥ ਹੈ ਕਿ ਪ੍ਰੋਗਰਾਮ ਵਿੱਚ ਸਿਰਫ਼ ਜ਼ਰੂਰੀ ਵਿਸ਼ੇਸ਼ਤਾਵਾਂ ਨੂੰ ਵਿਖਾਉਣਾ ਅਤੇ ਅੰਦਰੂਨੀ implementation ਨੂੰ ਛੁਪਾਉਣਾ।

Q. How Multilevel Inheritance is different from Multiple Inheritance? (Nov 22)

Ans. Multilevel inheritance involves a class derived from another derived class, while multiple inheritance means a class inherits from two or more base classes.

Multilevel inheritance ਵਿੱਚ ਇੱਕ ਕਲਾਸ, ਦੂਜੇ ਡਿਰਾਈਵਡ ਕਲਾਸ ਤੋਂ ਵਿਰਾਸਤ ਲੈਂਦੀ ਹੈ।

Multiple inheritance ਵਿੱਚ ਇੱਕ ਕਲਾਸ ਦੋ ਜਾਂ ਇਸ ਤੋਂ ਵੱਧ base classes ਤੋਂ ਵਿਰਾਸਤ ਲੈਂਦੀ ਹੈ।

Long Answer Questions:

Q. Explain characteristics of Object-Oriented Programming. (Nov 24)

Ans. **Object-Oriented Programming (OOP)** is a programming paradigm based on the concept of "objects", which can contain data and code. It helps in organizing complex programs, promoting reusability, scalability, and maintainability. The key characteristics of OOP are:

1. **Encapsulation:** It binds data and functions into a single unit called a class. It hides internal details and exposes only necessary parts through access modifiers (public, private, protected).
2. **Abstraction:** It allows programmers to focus on essential features without dealing with background complexity. Classes can expose required functionality while hiding implementation details.
3. **Inheritance:** It enables a class (derived class) to inherit properties and behaviors from another class (base class), allowing code reuse and hierarchical classification.
4. **Polymorphism:** It allows objects to take multiple forms. This can be achieved through function overloading, operator overloading, and virtual functions. It enables one interface to control access to different types of objects.
5. **Modularity:** Programs can be divided into smaller, independent units (classes and objects), which makes testing and maintenance easier.
6. **Reusability:** Once a class is written, it can be reused across programs or extended through inheritance without rewriting code.

OOP improves software design by making it more structured and close to real-world modeling.

Object-Oriented Programming (OOP) ਕੀ ਹੈ? Object-Oriented Programming (OOP) ਇੱਕ ਐਸਾ programming ਢੰਗ ਹੈ ਜੋ "objects" ਦੇ ਆਧਾਰ 'ਤੇ ਨਿਰਭਰ ਕਰਦਾ ਹੈ। ਇਹ objects ਵਿੱਚ data (ਡਾਟਾ) ਅਤੇ code (ਫੰਕਸ਼ਨ/ਤਰੀਕੇ) ਹੋ ਸਕਦੇ ਹਨ। OOP ਦੀ ਵਰਤੋਂ ਕਰਕੇ ਲੰਮੇ ਤੇ ਕਠਿਨ ਪ੍ਰੋਗਰਾਮ ਆਸਾਨੀ ਨਾਲ ਸੰਭਾਲੇ, ਦੁਬਾਰਾ ਵਰਤੇ ਅਤੇ ਵਧਾਏ ਜਾ ਸਕਦੇ ਹਨ।

OOP ਦੀਆਂ ਮੁੱਖ ਵਿਸ਼ੇਸ਼ਤਾਵਾਂ:

1. **Encapsulation (ਇੰਕੈਪਸੂਲੇਸ਼ਨ):** ਇਹ ਡਾਟਾ ਅਤੇ ਫੰਕਸ਼ਨਾਂ ਨੂੰ ਇੱਕ ਹੀ ਯੂਨਿਟ (class) ਵਿੱਚ ਜੋੜਦਾ ਹੈ। ਇਹ ਅੰਦਰੂਨੀ ਜਾਣਕਾਰੀ ਨੂੰ ਲੁਕਾ ਕੇ ਸਿਰਫ਼ ਜ਼ਰੂਰੀ ਹਿੱਸਾ ਹੀ ਉਪਲਬਧ ਕਰਵਾਉਂਦਾ ਹੈ। 📁 ਜਿਵੇਂ: public, private, protected access modifiers ਦੀ ਵਰਤੋਂ।

2. **Abstraction (ਐਬਸਟ੍ਰੈਕਸ਼ਨ):** ਇਹ ਵਰਤੋਂਕਾਰ ਨੂੰ ਸਿਰਫ਼ ਜ਼ਰੂਰੀ ਜਾਣਕਾਰੀ ਉਪਲਬਧ ਕਰਵਾਉਂਦਾ ਹੈ ਅਤੇ ਪਿਛਲੇ ਕੰਮ ਦੀ ਜਟਿਲਤਾ ਨੂੰ ਛੁਪਾਉਂਦਾ ਹੈ। 📁 ਜਿਵੇਂ: ਇੱਕ car ਦੀ driving functionality ਵੇਖੀ ਜਾ ਸਕਦੀ ਹੈ, ਪਰ engine ਦੀ ਅੰਦਰੂਨੀ ਬਣਾਵਟ ਨਹੀਂ।

3. Inheritance (ਇਨਹੈਰੀਟੈਂਸ): ਇਸਦੇ ਜ਼ਰੀਏ ਇੱਕ class (Derived class) ਦੂਜੇ class (Base class) ਤੋਂ ਗੁਣ (properties) ਅਤੇ ਵਿਵਹਾਰ (functions) ਲੈ ਸਕਦੀ ਹੈ। 📌 ਇਹ ਦੁਬਾਰਾ ਕੋਡ ਲਿਖਣ ਦੀ ਲੋੜ ਘਟਾਉਂਦਾ ਹੈ।

4. Polymorphism (ਪੋਲੀਮੋਰਫਿਜ਼ਮ): ਇਹ ਇੱਕ object ਨੂੰ ਕਈ ਰੂਪ ਵਿੱਚ ਵਰਤਣ ਦੀ ਸਹੂਲਤ ਦਿੰਦਾ ਹੈ। 📌 ਜਿਵੇਂ: Function overloading, Operator overloading, Virtual functions ਆਦਿ।

5. Modularity (ਮੋਡੀਊਲਰਿਟੀ): ਕਿਸੇ ਵੀ program ਨੂੰ ਛੋਟੇ ਛੋਟੇ ਹਿੱਸਿਆਂ (classes, objects) ਵਿੱਚ ਵੰਡਣਾ ਜੋ ਕਿ ਆਲਗ ਆਲਗ ਕੰਮ ਕਰਦੇ ਹਨ। 📌 ਇਹ testing ਅਤੇ maintenance ਆਸਾਨ ਬਣਾਉਂਦਾ ਹੈ।

6. Reusability (ਰੀਯੂਜੇਬਿਲਿਟੀ): ਜੇ class ਇੱਕ ਵਾਰ ਬਣਾਈ ਜਾਂਦੀ ਹੈ, ਉਹਨੂੰ ਬਾਅਦ ਵਿੱਚ ਕਈ programs ਵਿੱਚ ਦੁਬਾਰਾ ਵਰਤਿਆ ਜਾ ਸਕਦਾ ਹੈ ਜਾਂ inheritance ਰਾਹੀਂ ਵਧਾਇਆ ਜਾ ਸਕਦਾ ਹੈ।

ਨਤੀਜਾ (Conclusion): OOP programming ਨੂੰ ਵਿਅਵਸਥਿਤ, ਦੁਬਾਰਾ ਵਰਤਣਯੋਗ ਅਤੇ ਆਸਾਨੀ ਨਾਲ ਸੰਭਾਲਯੋਗ ਬਣਾਉਂਦਾ ਹੈ ਅਤੇ ਇਹ ਅਸਲ ਜਗਤ ਦੀ ਸਮਝ ਨੂੰ code ਵਿੱਚ ਲਿਆਉਂਦਾ ਹੈ।

Q. Explain constructor overloading. (Nov 24)

Ans. **Constructor overloading** in C++ is a feature that allows a class to have more than one constructor with different sets of parameters. It enables objects to be initialized in multiple ways using the same constructor name but with different argument lists.

Constructors are special member functions that are automatically called when an object of a class is created. Overloading constructors improves flexibility and reusability of code.

Example:

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

```
class Student {
    int id;
    string name;
```

```
public:
```

```
    // Default constructor
    Student() {
        id = 0;
        name = "Unknown";
    }
```

```
    // Parameterized constructor with one argument
    Student(int i) {
        id = i;
        name = "No Name";
    }
```

```
    // Parameterized constructor with two arguments
    Student(int i, string n) {
        id = i;
        name = n;
    }
```

```

void display() {
    cout << "ID: " << id << ", Name: " << name << endl;
}
};

int main() {
    Student s1;
    Student s2(101);
    Student s3(102, "Alice");

    s1.display();
    s2.display();
    s3.display();

    return 0;
}

```

In this example, the Student class has three constructors, each serving different initialization needs. The compiler differentiates them based on the number and type of parameters.

Constructor Overloading in C++ (ਕਨਸਟ੍ਰਕਟਰ ਓਵਰਲੋਡਿੰਗ)

C++ ਵਿੱਚ **Constructor Overloading** ਇੱਕ ਵਿਸ਼ੇਸ਼ਤਾ ਹੈ ਜੋ ਇੱਕ class ਨੂੰ ਵੱਖ-ਵੱਖ ਤਰੀਕਿਆਂ ਨਾਲ objects ਨੂੰ initialize ਕਰਨ ਦੀ ਸਹੂਲਤ ਦਿੰਦੀ ਹੈ। ਇਸ ਵਿੱਚ ਇੱਕੋ ਨਾਂ ਵਾਲੇ ਕਈ constructors ਹੋ ਸਕਦੇ ਹਨ ਪਰ ਉਹਨਾਂ ਦੇ parameters ਵੱਖ-ਵੱਖ ਹੁੰਦੇ ਹਨ।

Constructors ਉਹ special member functions ਹੁੰਦੇ ਹਨ ਜੋ object ਬਣਦੇ ਸਮੇਂ ਆਟੋਮੈਟਿਕ ਚੱਲਦੇ ਹਨ। Constructor overloading ਨਾਲ code ਵਿੱਚ flexibility ਅਤੇ reusability ਵਧਦੀ ਹੈ।

ਉਦਾਹਰਨ (Example):

```

#include <iostream>
using namespace std;

class Student {
    int id;
    string name;

public:
    // Default constructor
    Student() {
        id = 0;
        name = "Unknown";
    }

    // ਇੱਕ argument ਵਾਲਾ constructor
    Student(int i) {
        id = i;
        name = "No Name";
    }

    // ਦੋ arguments ਵਾਲਾ constructor
    Student(int i, string n) {

```

```

        id = i;
        name = n;
    }

    void display() {
        cout << "ID: " << id << ", Name: " << name << endl;
    }
};

int main() {
    Student s1;
    Student s2(101);
    Student s3(102, "Alice");

    s1.display();
    s2.display();
    s3.display();

    return 0;
}

```

ਸਮਝਾਓ (Explanation in Punjabi):

ਉਪਰੋਕਤ example ਵਿੱਚ Student class ਦੇ ਤਿੰਨ constructors ਹਨ:

1. **Default Constructor** – ਕੋਈ ਵੀ argument ਨਹੀਂ ਲੈਂਦਾ।
2. **Single Parameter Constructor** – ਸਿਰਫ id ਲੈਂਦਾ ਹੈ।
3. **Two Parameter Constructor** – id ਅਤੇ name ਦੋਵੇਂ ਲੈਂਦਾ ਹੈ।

C++ ਦਾ compiler ਇਹ constructors ਨੂੰ ਉਨ੍ਹਾਂ ਦੇ parameters ਦੀ ਗਿਣਤੀ ਅਤੇ type ਦੇ ਆਧਾਰ ਤੇ ਪਛਾਣਦਾ ਹੈ। ਇਹ feature object initialization ਨੂੰ ਆਸਾਨ ਅਤੇ flexible ਬਣਾਉਂਦਾ ਹੈ।

Q. What is a Copy Constructor and when is it called? (Nov 22)

Ans. A **copy constructor** is a special constructor in C++ used to create a new object as a **copy of an existing object**. It has the following general syntax:

ClassName(const ClassName &old_object);

It takes a reference to an object of the same class as an argument. The main purpose of a copy constructor is to perform a deep or customized copy of object data, especially when the class contains pointers or dynamically allocated memory.

When is a Copy Constructor Called?

A copy constructor is called in the following scenarios:

1. **When an object is initialized using another object:**

ClassName obj2 = obj1;

2. **When an object is passed by value to a function.**
3. **When a function returns an object by value.**
4. **When an object is explicitly copied.**

ClassName obj2(obj1);

Example:

```

class Demo {
public:
    int x;

```

```
Demo(int val) { x = val; }
Demo(const Demo &d) { x = d.x; } // Copy constructor
};
```

If you don't define a copy constructor, C++ provides a **default shallow copy**. However, for classes managing resources like files or dynamic memory, a user-defined copy constructor is necessary to avoid memory issues.

Copy Constructor in C++ (ਕਾਪੀ ਕਨਸਟ੍ਰਕਟਰ)

C++ ਵਿੱਚ Copy Constructor ਇੱਕ ਵਿਸ਼ੇਸ਼ constructor ਹੁੰਦਾ ਹੈ ਜੋ ਕਿਸੇ ਮੌਜੂਦਾ object ਦੀ ਨਕਲ ਕਰਕੇ ਨਵਾਂ object ਬਣਾਉਣ ਲਈ ਵਰਤਿਆ ਜਾਂਦਾ ਹੈ। ਇਸ ਦੀ ਆਮ syntax ਇਹ ਹੈ:

```
ClassName(const ClassName &old_object);
```

ਇਹ constructor ਇੱਕ reference ਲੈਂਦਾ ਹੈ ਉਸੇ ਹੀ class ਦੇ object ਦਾ। ਇਸ ਦਾ ਮੁੱਖ ਉਦੇਸ਼ dynamically allocated memory ਜਾਂ pointers ਵਾਲੇ objects ਦੀ deep copy ਕਰਨਾ ਹੁੰਦਾ ਹੈ।

Copy Constructor ਕਦੋਂ Call ਹੁੰਦਾ ਹੈ?

Copy constructor ਹੇਠ ਲਿਖੀਆਂ ਸਥਿਤੀਆਂ ਵਿੱਚ call ਹੁੰਦਾ ਹੈ:

1. Object Initialization ਨਾਲ:

```
ClassName obj2 = obj1;
```

2. Object ਨੂੰ Function ਨੂੰ pass ਕਰਨ ਸਮੇਂ (by value)
3. Function ਤੋਂ Object return ਕਰਨ ਸਮੇਂ (by value)
4. Explicit Copy ਕਰਦੇ ਹੋਏ:

```
ClassName obj2(obj1);
```

ਉਦਾਹਰਨ (Example):

```
class Demo {
public:
    int x;

    Demo(int val) { x = val; } // Parameterized constructor

    Demo(const Demo &d) { // Copy constructor
        x = d.x;
    }
};
```

ਸ਼ੈਲੋ ਕਾਪੀ (Shallow Copy) vs ਡੀਪ ਕਾਪੀ (Deep Copy):

ਜੇ ਤੁਸੀਂ Copy Constructor define ਨਹੀਂ ਕਰਦੇ, ਤਾਂ C++ ਇੱਕ default shallow copy ਬਣਾ ਦਿੰਦਾ ਹੈ। ਪਰ ਜਦੋਂ ਤੁਸੀਂ dynamically allocated memory ਜਾਂ pointers ਵਰਤ ਰਹੇ ਹੋ, ਤਾਂ ਇੱਕ user-defined copy constructor ਲਾਜ਼ਮੀ ਹੁੰਦਾ ਹੈ ਨਹੀਂ ਤਾਂ memory leak ਜਾਂ undefined behavior ਹੋ ਸਕਦਾ ਹੈ।

ਸਾਰ: Copy constructor object ਦੀ safe ਅਤੇ customized ਨਕਲ ਬਣਾਉਣ ਲਈ ਵਰਤਿਆ ਜਾਂਦਾ ਹੈ, ਖਾਸ ਕਰਕੇ ਜਦੋਂ object ਵਿੱਚ heap memory ਜਾਂ pointers ਹੁੰਦੇ ਹਨ।

Q. Explain the following: (Nov 20)

a) Polymorphism in C++ (Nov 20)

b) Classes in C++

c) Data Abstraction in C++

d) Inheritance in C++

Ans. **a) Polymorphism in C++:** Polymorphism means “many forms” and allows the same function name or operator to behave differently based on context. In C++, it is mainly of two types:

- **Compile-time polymorphism** (function/operator overloading)
 - **Run-time polymorphism** (using virtual functions and inheritance)
- For example, a function named display() can show different outputs based on the parameters passed.

b) Classes in C++: A **class** is a user-defined data type that serves as a blueprint for creating objects. It encapsulates data (variables) and functions (methods) into a single unit. Example:

```
class Student {  
    public:  
    int roll;  
    void show() { cout << roll; }  
};
```

c) Data Abstraction in C++: **Abstraction** means showing only essential features and hiding the internal details. In C++, abstraction is achieved using classes and access specifiers (private, public, protected). It helps in building secure and clean code by exposing only necessary parts of an object.

d) Inheritance in C++: **Inheritance** allows one class (child/derived) to acquire the properties and behavior of another class (parent/base). This promotes code reusability and supports the concept of hierarchical classification.

Example:

```
class Car : public Vehicle { };
```

a) C++ ਵਿੱਚ Polymorphism (ਪੋਲਿਮਾਰਫਿਜ਼ਮ): Polymorphism ਦਾ ਅਰਥ ਹੁੰਦਾ ਹੈ "ਕਈ ਰੂਪ"। ਇਹ ਇੱਕੋ ਹੀ function ਜਾਂ operator ਨੂੰ ਵੱਖ-ਵੱਖ context ਵਿੱਚ ਵੱਖ-ਵੱਖ ਢੰਗ ਨਾਲ ਵਰਤਣ ਦੀ ਆਗਿਆ ਦਿੰਦਾ ਹੈ।

C++ ਵਿੱਚ ਇਹ ਦੋ ਪ੍ਰਕਾਰਾਂ ਦੇ ਹੁੰਦੇ ਹਨ:

1. **Compile-time Polymorphism** – Function ਅਤੇ Operator Overloading ਰਾਹੀਂ।
2. **Run-time Polymorphism** – Virtual Functions ਅਤੇ Inheritance ਰਾਹੀਂ।

ਉਦਾਹਰਨ: ਇੱਕ display() function ਵੱਖ-ਵੱਖ arguments ਦੇ ਆਧਾਰ 'ਤੇ ਵੱਖ-ਵੱਖ output ਦੇ ਸਕਦਾ ਹੈ।

b) C++ ਵਿੱਚ Classes (ਕਲਾਸ): Class ਇੱਕ user-defined data type ਹੁੰਦੀ ਹੈ ਜੋ object ਬਣਾਉਣ ਲਈ blueprint ਵਜੋਂ ਕੰਮ ਕਰਦੀ ਹੈ। ਇਹ data (variables) ਅਤੇ functions (methods) ਨੂੰ ਇੱਕ ਇਕਾਈ ਵਿੱਚ ਬੰਨ੍ਹਦੀ ਹੈ।

ਉਦਾਹਰਨ:

```
class Student {  
    public:  
    int roll;  
    void show() { cout << roll; }  
};
```

c) C++ ਵਿੱਚ Data Abstraction (ਡਾਟਾ ਐਬਸਟਰੈਕਸ਼ਨ): Abstraction ਦਾ ਅਰਥ ਹੈ ਕਿ ਸਿਰਫ਼ ਜ਼ਰੂਰੀ ਜਾਣਕਾਰੀ ਦਿਖਾਈ ਜਾਵੇ ਅਤੇ ਅੰਦਰੂਨੀ ਜਟਿਲਤਾ ਨੂੰ ਲੁਕਾਇਆ ਜਾਵੇ।

C++ ਵਿੱਚ abstraction ਨੂੰ achieve ਕਰਨ ਲਈ **classes** ਅਤੇ **access specifiers** (private, public, protected) ਵਰਤੇ ਜਾਂਦੇ ਹਨ। ਇਹ code ਨੂੰ ਸੁਰੱਖਿਅਤ ਅਤੇ ਸਾਫ਼ ਬਣਾਉਂਦਾ ਹੈ।

d) C++ ਵਿੱਚ Inheritance (ਇਨਹੈਰਿਟੈਂਸ): Inheritance ਰਾਹੀਂ ਇੱਕ class (derived/child) ਦੂਜੇ class (base/parent) ਦੀਆਂ properties ਅਤੇ functions ਨੂੰ ਅਪਣਾਉਂਦੀ ਹੈ।

ਇਸ ਨਾਲ code reuse ਹੁੰਦੀ ਹੈ ਅਤੇ ਇੱਕ ਹਾਇਰਾਰਕੀ ਸਿਸਟਮ ਬਣਾਉਣ ਵਿੱਚ ਮਦਦ ਮਿਲਦੀ ਹੈ।

ਉਦਾਹਰਨ:

```
class Car : public Vehicle { };
```

Q. Define Classes and Objects. Explain the different method to accessing members of class using an example. (Nov 20)

Ans. A **class** is a user-defined data type in C++ that serves as a blueprint for creating **objects**. It groups related data members (variables) and member functions (methods) into a single unit.

An **object** is an instance of a class. It is created to use the properties and behavior defined by the class. Multiple objects can be created from a single class, each having its own copy of data members.

Example:

```
#include <iostream>
```

```
using namespace std;
```

```
class Student {
```

```
public:
```

```
    int roll;
```

```
    string name;
```

```
    void display() {
```

```
        cout << "Roll No: " << roll << ", Name: " << name << endl;
```

```
    }
```

```
};
```

```
int main() {
```

```
    Student s1; // creating object
```

```
    // Accessing class members
```

```
    s1.roll = 101;
```

```
    s1.name = "Rahul";
```

```
    s1.display(); // accessing function
```

```
    return 0;
```

```
}
```

Ways to Access Members of a Class:

1. Using dot operator (.) for normal objects (as shown above).

2. Using pointer to object with arrow operator (->):

```
Student *ptr = &s1;
```

```
ptr->display();
```

These methods allow controlled access to class members and support OOP principles like encapsulation and abstraction.

C++ ਵਿੱਚ Class ਤੇ Object ਕੀ ਹੁੰਦੇ ਹਨ?

Class C++ ਵਿੱਚ ਇੱਕ user-defined data type ਹੁੰਦੀ ਹੈ ਜੋ objects ਬਣਾਉਣ ਲਈ ਇੱਕ blueprint ਵਜੋਂ ਕੰਮ ਕਰਦੀ ਹੈ।

ਇਹ ਵਿੱਚ related data members (variables) ਅਤੇ member functions (methods) ਇਕੱਠੇ ਕੀਤੇ ਜਾਂਦੇ ਹਨ।

Object ਇੱਕ class ਦਾ instance ਹੁੰਦਾ ਹੈ। Object class ਵਿੱਚ define ਕੀਤੀਆਂ properties ਅਤੇ behaviors (ਫੰਕਸ਼ਨ) ਨੂੰ ਵਰਤਣ ਲਈ ਬਣਾਇਆ ਜਾਂਦਾ ਹੈ। ਇੱਕੋ class ਤੋਂ ਕਈ objects ਬਣਾਏ ਜਾ ਸਕਦੇ ਹਨ ਅਤੇ ਹਰ object ਦੀ ਆਪਣੀ ਅਲਗ copy ਹੁੰਦੀ ਹੈ।

ਉਦਾਹਰਨ:

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

class Student {
public:
    int roll;
    string name;

    void display() {
        cout << "Roll No: " << roll << ", Name: " << name << endl;
    }
};

int main() {
    Student s1; // object ਬਣਾਇਆ

    // Class ਦੇ members ਨੂੰ access ਕਰਨਾ
    s1.roll = 101;
    s1.name = "Rahul";
    s1.display(); // function call

    return 0;
}
```

Class Members ਨੂੰ Access ਕਰਨ ਦੇ ਤਰੀਕੇ:

1. Dot Operator (.) – ਸਧਾਰਣ object ਰਾਹੀਂ access:

```
s1.display();
```

2. Pointer ਰਾਹੀਂ Arrow Operator (->) – ਜਦੋਂ ਤੁਸੀਂ object ਦਾ pointer ਬਣਾਉਂ:

```
Student *ptr = &s1;
ptr->display();
```

ਇਹ ਤਰੀਕੇ C++ ਵਿੱਚ Encapsulation ਅਤੇ Abstraction ਵਰਗੀਆਂ OOP concepts ਨੂੰ follow ਕਰਦੇ ਹਨ, ਜੋ code ਨੂੰ structure ਅਤੇ security ਦਿੰਦੇ ਹਨ।

ਜੇ ਤੁਸੀਂ ਹੋਰ OOP concepts ਦੀ ਵੀ ਪੰਜਾਬੀ ਵਿੱਚ ਵਿਆਖਿਆ ਚਾਹੁੰਦੇ ਹੋ ਤਾਂ ਦੱਸੋ, ਮੈਂ ਨੋਟਸ ਤਿਆਰ ਕਰ ਦਿਆਂ।

Q. Write a program to overload ++ and-- operator to increase and decrease the value of class data members. (Nov 24)

Ans. Operator overloading in C++ allows you to redefine the meaning of operators for user-defined types (like classes). Below is an example that demonstrates overloading of the ++ and-- operators to modify a class data member.

✓ Code:

```
#include <iostream>
```

```

using namespace std;

class Counter {
private:
    int value;

public:
    Counter() { value = 0; }

    // Overloading ++ (prefix)
    void operator++() {
        ++value;
    }

    // Overloading -- (prefix)
    void operator--() {
        --value;
    }

    void display() {
        cout << "Value: " << value << endl;
    }
};

int main() {
    Counter c;

    cout << "Initial ";
    c.display();

    ++c; // calls operator++
    cout << "After ++ ";
    c.display();

    --c; // calls operator--
    cout << "After-- ";
    c.display();

    return 0;
}

```

✓ Explanation:

- operator++() is used to increment the value.
- operator--() is used to decrement the value.
- These are **prefix** forms; you can also define **postfix** versions by using a dummy int parameter.
- This demonstrates encapsulation and operator overloading concepts in C++.

✓ C++ ਵਿੱਚ Operator Overloading (ਓਪਰੇਟਰ ਓਵਰਲੋਡਿੰਗ)

C++ ਵਿੱਚ operator overloading ਦਾ ਮਤਲੱਬ ਹੈ ਕਿ ਤੁਸੀਂ ਕਿਸੇ user-defined type (ਜਿਵੇਂ ਕਿ class) ਲਈ operators ਦੀ ਵਿਵਹਾਰਤਾ ਨੂੰ ਦੁਬਾਰਾ ਭਰਾ ਸਕਦੇ ਹੋ।

ਹੇਠਾਂ ਦਿੱਤਾ ਉਦਾਹਰਨ ++ ਅਤੇ-- ਓਪਰੇਟਰ ਨੂੰ overload ਕਰਕੇ class ਦੇ member variable ਨੂੰ ਵਧਾਉਣ ਜਾਂ ਘਟਾਉਣ ਦਾ ਤਰੀਕਾ ਦਿਖਾਉਂਦਾ ਹੈ।

**ਕੋਡ:**

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

class Counter {
private:
    int value;

public:
    Counter() { value = 0; }

    // ++ ਓਪਰੇਟਰ ਦੀ overloading (prefix)
    void operator++() {
        ++value;
    }

    // -- ਓਪਰੇਟਰ ਦੀ overloading (prefix)
    void operator--() {
        --value;
    }

    void display() {
        cout << "Value: " << value << endl;
    }
};

int main() {
    Counter c;

    cout << "Initial ";
    c.display();

    ++c; // operator++() call ਹੁੰਦੀ ਹੈ
    cout << "After ++ ";
    c.display();

    --c; // operator--() call ਹੁੰਦੀ ਹੈ
    cout << "After-- ";
    c.display();

    return 0;
}
```

**ਵਿਆਖਿਆ (Explanation in Punjabi):**

- operator++() function **value** ਨੂੰ 1 ਵਧਾਉਂਦਾ ਹੈ।
- operator--() function **value** ਨੂੰ 1 ਘਟਾਉਂਦਾ ਹੈ।
- ਇਹ ਦੋਵੇਂ **prefix form** ਹਨ (ਜਿਵੇਂ ++x, --x)।

- ਜੇ ਤੁਸੀਂ postfix form ($x++$, $x--$) ਬਣਾਉਣਾ ਚਾਹੁੰਦੇ ਹੋ ਤਾਂ ਤੁਸੀਂ function ਵਿੱਚ dummy int parameter ਵਰਤ ਸਕਦੇ ਹੋ।

 ਇਹ ਕੋਡ C++ ਦੇ Encapsulation ਅਤੇ Operator Overloading ਦੇ concepts ਨੂੰ ਦਰਸਾਉਂਦਾ ਹੈ।

Unit 4: Inheritance and Polymorphism

Short Answer Questions:

Q. What are public and private keywords? (Nov 20)

Ans. public members are accessible from anywhere in the program, while private members can only be accessed within the class itself.

public ਮੈਂਬਰ ਕਿਤੇ ਵੀ program ਵਿੱਚ ਐਕਸੈੱਸ ਕੀਤੇ ਜਾ ਸਕਦੇ ਹਨ, ਜਦਕਿ private ਮੈਂਬਰ ਸਿਰਫ਼ ਉਸੀ ਕਲਾਸ ਵਿੱਚ ਹੀ ਐਕਸੈੱਸ ਕੀਤੇ ਜਾ ਸਕਦੇ ਹਨ।

Q. Differentiate between private and protected class members. (Nov 20)

Ans. private members are accessible only within the class, while protected members are accessible within the class and its derived classes.

private ਮੈਂਬਰ ਸਿਰਫ਼ ਕਲਾਸ ਦੇ ਅੰਦਰ ਹੀ ਐਕਸੈੱਸ ਕੀਤੇ ਜਾ ਸਕਦੇ ਹਨ, ਜਦਕਿ protected ਮੈਂਬਰ ਕਲਾਸ ਅਤੇ ਉਸ ਦੀਆਂ derived classes ਵਿੱਚ ਐਕਸੈੱਸ ਕੀਤੇ ਜਾ ਸਕਦੇ ਹਨ।

Q. Discuss any two disadvantages of Multiple inheritance. (Nov 20)

Ans. Multiple inheritance can cause ambiguity (e.g., diamond problem) and increases code complexity, making maintenance harder.

Ambiguity (ਅਸਪਸ਼ਟਤਾ) ਪੈਦਾ ਹੋ ਸਕਦੀ ਹੈ, ਜਿਵੇਂ ਕਿ diamond problem। ਕੋਡ ਜ਼ਿਆਦਾ complex ਹੋ ਜਾਂਦਾ ਹੈ, ਜਿਸ ਨਾਲ maintenance ਮੁਸ਼ਕਲ ਹੋ ਜਾਂਦੀ ਹੈ।

Q. Virtual base class (Nov 24)

Ans. A virtual base class prevents multiple copies of a base class in a multiple inheritance hierarchy, resolving ambiguity.

Virtual base class ਮਲਟੀਪਲ ਇਨਹੈਰੀਟੈਂਸ ਵਿੱਚ base class ਦੀ duplicate copies ਬਣਨ ਤੋਂ ਰੋਕਦੀ ਹੈ, ਜਿਸ ਨਾਲ ambiguity ਨੂੰ ਦੂਰ ਕੀਤਾ ਜਾਂਦਾ ਹੈ।

Q. What is an exception in C++? (Nov 22)

Ans. An exception is a runtime error-handling mechanism that allows the program to catch and manage unexpected errors using try, catch, and throw.

Exception ਇੱਕ runtime error-handling mechanism ਹੈ ਜੋ try, catch ਅਤੇ throw keywords ਦੀ ਵਰਤੋਂ ਕਰਕੇ ਅਣਪਿੱਛਲੇ errors ਨੂੰ handle ਕਰਦਾ ਹੈ।

Q. How do you allocate and deallocate memory in C++? (Nov 22)

Ans. Use new to dynamically allocate memory and delete to free it when no longer needed.

new keyword ਨਾਲ dynamically memory allocate ਕੀਤੀ ਜਾਂਦੀ ਹੈ ਅਤੇ delete ਨਾਲ ਉਹ memory ਖਾਲੀ ਕੀਤੀ ਜਾਂਦੀ ਹੈ ਜਦੋਂ ਉਹ ਲੋੜੀਂਦੀ ਨਾ ਰਹੇ।

Q. New (Nov 24)

ans. new is a dynamic memory operator in C++ used to allocate memory for variables or objects at runtime.

new ਇੱਕ dynamic memory operator ਹੈ ਜੋ runtime ਤੇ variables ਜਾਂ objects ਲਈ memory allocate ਕਰਨ ਲਈ ਵਰਤਿਆ ਜਾਂਦਾ ਹੈ।

Long Answer Questions:

Q. What is runtime polymorphism in C++? Write a program to illustrate it. (Nov 22)

Ans. Runtime Polymorphism in C++ (in 200 Words):

Runtime polymorphism in C++ is a form of polymorphism that occurs during program execution. It is achieved using **inheritance** and **virtual functions**. This allows a **base class pointer** to call functions from **derived class objects** at runtime, enabling dynamic method dispatch.

The primary requirement is to declare a function as virtual in the base class and override it in the derived class. This allows C++ to decide at **runtime** which function to invoke, based on the actual object type.

✅ **Program to Illustrate Runtime Polymorphism:**

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

class Animal {
public:
    virtual void sound() {
        cout << "Animal makes a sound" << endl;
    }
};

class Dog : public Animal {
public:
    void sound() override {
        cout << "Dog barks" << endl;
    }
};

class Cat : public Animal {
public:
    void sound() override {
        cout << "Cat meows" << endl;
    }
};

int main() {
    Animal* a; // base class pointer
    Dog d;
    Cat c;

    a = &d;
    a->sound(); // Calls Dog's sound()

    a = &c;
    a->sound(); // Calls Cat's sound()

    return 0;
}
```

✅ **Output:**

Dog barks
Cat meows

✅ **Conclusion:**

Runtime polymorphism enhances flexibility and reusability by allowing objects to behave differently depending on their actual class at runtime.

✅ **C++ ਵਿੱਚ Runtime Polymorphism (200 ਸ਼ਬਦਾਂ ਵਿੱਚ)**

C++ ਵਿੱਚ Runtime Polymorphism ਉਹ ਵਿਵਹਾਰ ਹੈ ਜੋ ਕੰਪਾਇਲ ਹੋਣ ਤੋਂ ਬਾਅਦ, ਚਲਾਉਣ ਦੇ ਸਮੇਂ (runtime) ਤੇ ਤੈਅ ਹੁੰਦਾ ਹੈ। ਇਹ ਇਨਹੈਰਿਟੈਂਸ (Inheritance) ਅਤੇ ਵਰਚੁਅਲ ਫੰਕਸ਼ਨਾਂ (Virtual Functions) ਰਾਹੀਂ ਹਾਸਲ ਕੀਤਾ ਜਾਂਦਾ ਹੈ। ਇਸਦੇ ਜ਼ਰੀਏ, ਇੱਕ base class ਦਾ pointer ਜਾਂ reference, derived class ਦੀ object ਨੂੰ point ਕਰ ਸਕਦਾ ਹੈ ਅਤੇ ਉਸ object ਦੀ function definition ਨੂੰ runtime 'ਤੇ ਚਲਾਉਂਦਾ ਹੈ।

★ ਜ਼ਰੂਰੀ ਗੱਲ:

- Base class ਵਿੱਚ function ਨੂੰ virtual ਘੋਸ਼ਿਤ ਕੀਤਾ ਜਾਂਦਾ ਹੈ।
- Derived class ਵਿੱਚ ਉਸ function ਨੂੰ override ਕੀਤਾ ਜਾਂਦਾ ਹੈ।
- ਇਹ mechanism dynamic dispatch ਕਹੀ ਜਾਂਦੀ ਹੈ।

✓ ਉਦਾਹਰਨ – Runtime Polymorphism ਦਾ ਕੋਡ:

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

class Animal {
public:
    virtual void sound() {
        cout << "Animal makes a sound" << endl;
    }
};

class Dog : public Animal {
public:
    void sound() override {
        cout << "Dog barks" << endl;
    }
};

class Cat : public Animal {
public:
    void sound() override {
        cout << "Cat meows" << endl;
    }
};

int main() {
    Animal* a;    // base class pointer
    Dog d;
    Cat c;

    a = &d;
    a->sound();   // Dog ਦੀ sound() call ਹੁੰਦੀ ਹੈ

    a = &c;
    a->sound();   // Cat ਦੀ sound() call ਹੁੰਦੀ ਹੈ

    return 0;
}
```


✓ ਆਉਟਪੁਟ (Output):

Dog barks
Cat meows

✓ **ਨਤੀਜਾ (Conclusion):** Runtime polymorphism C++ ਵਿੱਚ ਇੱਕ ਸ਼ਕਤੀਸ਼ਾਲੀ ਵਿਸ਼ੇਸ਼ਤਾ ਹੈ ਜੋ code ਨੂੰ **ਫਲੈਕਸੀਬਲ**, reuseable ਅਤੇ **ਵਧੀਕ ਮਾਡਿਊਲਰ** ਬਣਾਉਂਦੀ ਹੈ। ਇਹ allow ਕਰਦੀ ਹੈ ਕਿ objects ਆਪਣੇ ਅਸਲ type ਅਨੁਸਾਰ runtime 'ਤੇ ਅਲੱਗ ਵਿਵਹਾਰ ਕਰ ਸਕਣ।

Q. What are the various types of access specifiers used in C++? (Nov 24)

Ans. Access specifiers in C++ are keywords used to set the **accessibility** or **visibility** of class members (variables and functions). They help implement **encapsulation** by controlling how class members are accessed from outside the class. There are **three main access specifiers** in C++:

1. Public:

- Members declared as public are accessible **from anywhere** in the program using the object of the class.
- Used when functions or variables need to be accessed freely.

Example:

```
class Demo {  
public:  
    int x; // accessible from outside  
};
```

2. Private:

- Members declared as private can only be accessed **within the class itself**.
- They are not accessible directly from outside the class.
- Used to protect sensitive data.

Example:

```
class Demo {  
private:  
    int x; // not accessible directly outside the class  
};
```

3. Protected:

- Members are accessible within the class and **in derived (child) classes**.
- Not accessible directly from outside the class or from unrelated classes.

Example:

```
class Base {  
protected:  
    int x; // accessible in derived class  
};
```

Access specifiers are crucial for building **secure and modular** object-oriented applications in C++.

C++ ਵਿੱਚ Access Specifiers ਕੀ ਹੁੰਦੇ ਹਨ?

Access specifiers C++ ਵਿੱਚ ਕੁੰਜੀ-ਸ਼ਬਦ (keywords) ਹੁੰਦੇ ਹਨ ਜੋ class ਦੇ ਮੈਂਬਰਾਂ (variables ਅਤੇ functions) ਦੀ **ਪਹੁੰਚ (accessibility)** ਜਾਂ **ਦਿੱਖ (visibility)** ਨੂੰ ਨਿਰਧਾਰਤ ਕਰਦੇ ਹਨ। ਇਹ encapsulation ਲਾਗੂ ਕਰਨ ਵਿੱਚ ਮਦਦ ਕਰਦੇ ਹਨ ਜਿਸ ਨਾਲ ਬਾਹਰਲੇ ਕੋਡ ਨੂੰ ਕਿਵੇਂ ਅਤੇ ਕਿੱਥੋਂ class ਦੇ ਮੈਂਬਰਾਂ ਤੱਕ ਪਹੁੰਚ ਮਿਲੇ, ਇਹ ਕੰਟਰੋਲ ਹੁੰਦਾ ਹੈ।

C++ ਵਿੱਚ ਤਿੰਨ ਪ੍ਰਮੁੱਖ access specifiers ਹੁੰਦੇ ਹਨ:

1. Public:

- Public ਮੈਂਬਰਾਂ ਤੱਕ class ਦੀ ਕਿਸੇ ਵੀ object ਰਾਹੀਂ ਕਿਤੇ ਵੀ ਪਹੁੰਚ ਹੋ ਸਕਦੀ ਹੈ।
- ਜਦੋਂ ਤੁਹਾਨੂੰ functions ਜਾਂ variables ਨੂੰ ਮੁਫਤ ਤੌਰ ਤੇ ਕਿਤੇ ਵੀ access ਕਰਵਾਉਣਾ ਹੋਵੇ, ਤਦ ਇਹ ਵਰਤਿਆ ਜਾਂਦਾ ਹੈ।

ਉਦਾਹਰਨ:

```
class Demo {  
public:  
    int x; // ਬਾਹਰੋਂ ਸਿੱਧਾ access ਕੀਤਾ ਜਾ ਸਕਦਾ ਹੈ  
};
```

2. Private:

- Private ਮੈਂਬਰ ਸਿਰਫ਼ ਉਸੀ class ਦੇ ਅੰਦਰੋਂ ਹੀ access ਹੋ ਸਕਦੇ ਹਨ।
- ਬਾਹਰਲੇ ਕੋਡ ਤੋਂ ਇਹ ਸਿੱਧਾ access ਨਹੀਂ ਕੀਤੇ ਜਾ ਸਕਦੇ।
- ਇਹ ਸੰਵੇਦਨਸ਼ੀਲ (sensitive) ਡਾਟਾ ਦੀ ਰੱਖਿਆ ਲਈ ਵਰਤਿਆ ਜਾਂਦਾ ਹੈ।

ਉਦਾਹਰਨ:

```
class Demo {  
private:  
    int x; // ਬਾਹਰੋਂ ਸਿੱਧਾ access ਨਹੀਂ ਕੀਤਾ ਜਾ ਸਕਦਾ  
};
```

3. Protected:

- Protected ਮੈਂਬਰ class ਦੇ ਅੰਦਰ ਅਤੇ derived (ਬੱਚੇ) classes ਵਿੱਚ access ਹੋ ਸਕਦੇ ਹਨ।
- ਬਾਹਰਲੇ ਕੋਈ ਵੀ unrelated ਕੋਡ ਇਸ ਤਰ੍ਹਾਂ ਦੀ access ਨਹੀਂ ਕਰ ਸਕਦਾ।

ਉਦਾਹਰਨ:

```
cpp  
CopyEdit  
class Base {  
protected:  
    int x; // derived class ਵਿੱਚ access ਕੀਤਾ ਜਾ ਸਕਦਾ ਹੈ  
};
```

ਨਤੀਜਾ:

Access specifiers C++ ਵਿੱਚ ਸੁਰੱਖਿਅਤ ਅਤੇ ਮਾਡਿਊਲਰ object-oriented applications ਬਣਾਉਣ ਲਈ ਬਹੁਤ ਜ਼ਰੂਰੀ ਹਨ। ਇਹ encapsulation ਅਤੇ data hiding ਨੂੰ ਮਜ਼ਬੂਤ ਕਰਦੇ ਹਨ।

Q. Explain various types of inheritance that we can perform in C++. Give examples. (Nov 24)

Ans. Inheritance allows a new class (derived class) to acquire properties and behaviors of an existing class (base class), promoting code reuse. C++ supports several types of inheritance:

-
1. **Single Inheritance:** One derived class inherits from one base class.

Example:

```
class Animal {  
public:  
    void eat() { cout << "Eating\n"; }  
};
```

```
class Dog : public Animal {
public:
    void bark() { cout << "Barking\n"; }
};
```

2. **Multiple Inheritance:** A derived class inherits from more than one base class.

Example:

```
class Printer {
public:
    void print() { cout << "Printing\n"; }
};

class Scanner {
public:
    void scan() { cout << "Scanning\n"; }
};

class Copier : public Printer, public Scanner { };
```

3. **Multilevel Inheritance:** A derived class inherits from another derived class, forming a chain.

Example:

```
class Animal {
public:
    void eat() { cout << "Eating\n"; }
};

class Dog : public Animal {
public:
    void bark() { cout << "Barking\n"; }
};

class Puppy : public Dog {
public:
    void weep() { cout << "Weeping\n"; }
};
```

4. **Hierarchical Inheritance:** Multiple derived classes inherit from a single base class.

Example:

```
class Animal { };
class Dog : public Animal { };
class Cat : public Animal { };
```

5. **Hybrid Inheritance:** Combination of two or more types of inheritance.

Inheritance supports better code organization and reusability in C++.

ਵਿਰਾਸਤ (Inheritance) ਕੀ ਹੈ?

Inheritance ਇੱਕ ਨਵੀਂ class (derived class) ਨੂੰ ਮੌਜੂਦਾ class (base class) ਦੀਆਂ ਖਾਸੀਅਤਾਂ ਅਤੇ ਕਰਤੱਬਾਂ ਨੂੰ ਹਾਸਲ ਕਰਨ ਦੀ ਸਹੂਲਤ ਦਿੰਦਾ ਹੈ, ਜਿਸ ਨਾਲ ਕੋਡ ਦੁਹਰਾਅ ਘਟਦਾ ਹੈ ਅਤੇ ਵਰਤੋਂ ਵਧਦੀ ਹੈ। C++ ਵਿੱਚ ਵੱਖ-ਵੱਖ ਕਿਸਮਾਂ ਦੀ inheritance ਨੂੰ ਸਹਾਰਾ ਮਿਲਦਾ ਹੈ:

1. Single Inheritance (ਸਿੰਗਲ ਵਿਰਾਸਤ): ਇੱਕ derived class ਇੱਕ base class ਤੋਂ ਵਿਰਾਸਤ ਲੈਂਦੀ ਹੈ।

ਉਦਾਹਰਨ:

```
class Animal {
public:
    void eat() { cout << "Eating\n"; }
};

class Dog : public Animal {
public:
    void bark() { cout << "Barking\n"; }
};
```

2. Multiple Inheritance (ਬਹੁ-ਵਿਰਾਸਤ): ਇੱਕ derived class ਇੱਕ ਤੋਂ ਵੱਧ base classes ਤੋਂ ਵਿਰਾਸਤ ਲੈਂਦੀ ਹੈ।

ਉਦਾਹਰਨ:

```
class Printer {
public:
    void print() { cout << "Printing\n"; }
};

class Scanner {
public:
    void scan() { cout << "Scanning\n"; }
};

class Copier : public Printer, public Scanner { };
```

3. Multilevel Inheritance (ਬਹੁ-ਸਤਹੀ ਵਿਰਾਸਤ): ਇੱਕ derived class, ਹੋਰ derived class ਤੋਂ ਵਿਰਾਸਤ ਲੈਂਦੀ ਹੈ, ਜਿਸ ਨਾਲ ਕੜੀ ਬਣਦੀ ਹੈ।

ਉਦਾਹਰਨ:

```
class Animal {
public:
    void eat() { cout << "Eating\n"; }
};

class Dog : public Animal {
public:
    void bark() { cout << "Barking\n"; }
};

class Puppy : public Dog {
public:
    void weep() { cout << "Weeping\n"; }
};
```

4. Hierarchical Inheritance (ਹਾਇਰਾਰਕੀਕਲ ਵਿਰਾਸਤ): ਕਈ derived classes ਇੱਕ ਹੀ base class ਤੋਂ ਵਿਰਾਸਤ ਲੈਂਦੀਆਂ ਹਨ।

ਉਦਾਹਰਨ:

```
class Animal { };
class Dog : public Animal { };
```

```
class Cat : public Animal { };
```

5. Hybrid Inheritance (ਮਿਲੀ ਜੁਲੀ ਵਿਰਾਸਤ):

ਦੇ ਜਾਂ ਵੱਧ inheritance ਦੇ ਤਰੀਕਿਆਂ ਦਾ ਮਿਲਾਪ।

ਨਤੀਜਾ:

Inheritance C++ ਵਿੱਚ ਕੋਡ ਨੂੰ ਵਧੀਆ ਢੰਗ ਨਾਲ ਆਯੋਜਿਤ ਅਤੇ ਦੁਹਰਾਅ ਯੋਗ ਬਣਾਉਣ ਵਿੱਚ ਮਦਦ ਕਰਦਾ ਹੈ।

Q. How can you convert an integer to string in C++? Write a program. (Nov 22)

Ans. In C++, an integer can be converted to a string in multiple ways. The most common and simplest method is to use the **to_string()** function provided by the C++ Standard Library (<string> header). This function takes an integer (or other numeric types) and returns its string representation.

Example Program:

```
#include <iostream>
#include <string> // Required for std::to_string

using namespace std;

int main() {
    int num = 12345;

    // Convert integer to string using to_string()
    string str = to_string(num);

    cout << "Integer: " << num << endl;
    cout << "String: " << str << endl;

    return 0;
}
```

Explanation:

- The `to_string()` function converts the integer `num` into a string `str`.
 - This string can then be used like any other string object for operations such as concatenation, output, or manipulation.
 - Before C++11, programmers often used `stringstream` from <sstream> for conversion, but `to_string()` is simpler and more efficient.
-

Output:

Integer: 12345

String: 12345

This method ensures clean and efficient conversion of integers to strings in modern C++.

C++ ਵਿੱਚ ਇੰਟੀਜਰ ਨੂੰ ਸਟਰਿੰਗ ਵਿੱਚ ਕਿਵੇਂ ਬਦਲਿਆ ਜਾ ਸਕਦਾ ਹੈ?

C++ ਵਿੱਚ ਇੱਕ ਇੰਟੀਜਰ ਨੂੰ ਸਟਰਿੰਗ ਵਿੱਚ ਕਈ ਤਰੀਕਿਆਂ ਨਾਲ ਬਦਲਿਆ ਜਾ ਸਕਦਾ ਹੈ। ਸਭ ਤੋਂ ਆਮ ਅਤੇ ਆਸਾਨ ਤਰੀਕਾ ਹੈ `to_string()` ਫੰਕਸ਼ਨ ਦੀ ਵਰਤੋਂ ਕਰਨੀ, ਜੋ C++ Standard Library (<string> ਹੇਡਰ) ਵਿੱਚ ਮਿਲਦਾ ਹੈ। ਇਹ ਫੰਕਸ਼ਨ ਕਿਸੇ ਇੰਟੀਜਰ ਜਾਂ ਹੋਰ ਨੰਬਰਿਕ ਕਿਸਮ ਨੂੰ ਲੈਂਦਾ ਹੈ ਅਤੇ ਉਸਦੀ ਸਟਰਿੰਗ ਰੂਪ ਵਿੱਚ ਵਾਪਸ ਕਰਦਾ ਹੈ।

ਉਦਾਹਰਨ ਦਾ ਕੋਡ:

```
#include <iostream>
```

```
#include <string> // std::to_string ਲਈ ਲੋੜੀਂਦਾ

using namespace std;

int main() {
    int num = 12345;

    // to_string() ਦੀ ਵਰਤੋਂ ਕਰਕੇ ਇੰਟੀਜਰ ਨੂੰ ਸਟਰਿੰਗ ਵਿੱਚ ਬਦਲੋ
    string str = to_string(num);

    cout << "Integer: " << num << endl;
    cout << "String: " << str << endl;

    return 0;
}
```

ਵਿਆਖਿਆ:

- to_string() ਫੰਕਸ਼ਨ ਇੰਟੀਜਰ num ਨੂੰ ਸਟਰਿੰਗ str ਵਿੱਚ ਬਦਲ ਦਿੰਦਾ ਹੈ।
- ਇਹ ਸਟਰਿੰਗ ਫਿਰ ਕਿਸੇ ਵੀ ਸਧਾਰਣ ਸਟਰਿੰਗ ਵਾਂਗ ਵਰਤੀ ਜਾ ਸਕਦੀ ਹੈ, ਜਿਵੇਂ ਕਿ ਕਨਕੈਟੇਨੇਸ਼ਨ, ਪ੍ਰਿੰਟਿੰਗ ਜਾਂ ਹੋਰ ਮੈਨਿਪੂਲੇਸ਼ਨ।
- C++11 ਤੋਂ ਪਹਿਲਾਂ, ਇਹ ਕੰਮ stringstream ਦੀ ਵਰਤੋਂ ਕਰਕੇ ਕੀਤਾ ਜਾਂਦਾ ਸੀ, ਪਰ to_string() ਵਰਤੋਂ ਵਿੱਚ ਆਸਾਨ ਅਤੇ ਤੇਜ਼ ਹੈ।

ਆਉਟਪੁੱਟ:

```
Integer: 12345
String: 12345
```

ਇਹ ਤਰੀਕਾ ਆਧੁਨਿਕ C++ ਵਿੱਚ ਇੰਟੀਜਰ ਤੋਂ ਸਟਰਿੰਗ ਤਬਦੀਲੀ ਲਈ ਸੁਭੰਤਰ ਅਤੇ ਪ੍ਰਭਾਵਸ਼ਾਲੀ ਹੈ।

Q. What is Exception Handling? Does C++ support Exception Handling? Comment on C++ standard exceptions. (Nov 22)

Ans. **Exception handling** is a mechanism to handle runtime errors or unusual conditions in a controlled way, allowing a program to continue or terminate gracefully without crashing. It separates error-handling code from regular code, improving clarity and robustness.

In C++, exception handling is fully supported using three keywords:

- try — defines a block of code to monitor for exceptions.
- throw — raises an exception.
- catch — handles the exception thrown.

How It Works:

When an exception occurs inside the try block, control is transferred to the matching catch block that handles the exception. If no matching catch block is found, the program terminates.

C++ Standard Exceptions:

C++ provides a standard exception hierarchy in the <exception> header, with std::exception as the base class. Common standard exceptions include:

- std::runtime_error — errors detected during runtime.
- std::logic_error — errors in the program logic.
- std::out_of_range — accessing invalid array or container indices.

- `std::invalid_argument` — invalid function arguments.
- `std::bad_alloc` — memory allocation failure.

These standard exceptions help developers catch and handle common error scenarios efficiently, promoting better error management.

In summary, C++ supports structured exception handling that improves program reliability by managing errors without abrupt termination.

ਐਕਸਪੈਸ਼ਨ ਹੈਂਡਲਿੰਗ ਕੀ ਹੈ?

ਐਕਸਪੈਸ਼ਨ ਹੈਂਡਲਿੰਗ ਇੱਕ ਤਰੀਕਾ ਹੈ ਜਿਸ ਨਾਲ ਰਨਟਾਈਮ ਦੌਰਾਨ ਹੋਣ ਵਾਲੀਆਂ ਗਲਤੀਆਂ ਜਾਂ ਅਸਧਾਰਣ ਸਥਿਤੀਆਂ ਨੂੰ ਕੰਟਰੋਲਡ ਢੰਗ ਨਾਲ ਸੰਭਾਲਿਆ ਜਾ ਸਕਦਾ ਹੈ। ਇਸ ਤਰੀਕੇ ਨਾਲ ਪ੍ਰੋਗਰਾਮ ਨੂੰ ਬਿਨਾਂ ਕਰੈਸ਼ ਹੋਏ ਜਾਰੀ ਰੱਖਣਾ ਜਾਂ ਸੁਚੱਜੇ ਤਰੀਕੇ ਨਾਲ ਬੰਦ ਕਰਨਾ ਮੁਮਕਿਨ ਹੁੰਦਾ ਹੈ। ਇਹ ਗਲਤੀ ਸੰਭਾਲਣ ਵਾਲੇ ਕੋਡ ਨੂੰ ਆਮ ਕੋਡ ਤੋਂ ਵੱਖਰਾ ਕਰਕੇ ਕੋਡ ਦੀ ਸਪਸ਼ਟਤਾ ਅਤੇ ਮਜ਼ਬੂਤੀ ਵਧਾਉਂਦਾ ਹੈ।

C++ ਵਿੱਚ ਐਕਸਪੈਸ਼ਨ ਹੈਂਡਲਿੰਗ

C++ ਵਿੱਚ ਐਕਸਪੈਸ਼ਨ ਹੈਂਡਲਿੰਗ ਤਿੰਨ ਮੁੱਖ ਕੀਵਰਡਾਂ ਦੀ ਵਰਤੋਂ ਨਾਲ ਕੀਤੀ ਜਾਂਦੀ ਹੈ:

- `try` — ਉਹ ਕੋਡ ਬਲਾਕ ਜੋ ਐਕਸਪੈਸ਼ਨਾਂ ਲਈ ਮਾਨੀਟਰ ਕੀਤਾ ਜਾਂਦਾ ਹੈ।
- `throw` — ਐਕਸਪੈਸ਼ਨ ਨੂੰ ਉੱਠਾਉਂਦਾ ਹੈ।
- `catch` — ਉੱਠਾਏ ਗਏ ਐਕਸਪੈਸ਼ਨ ਨੂੰ ਹੈਂਡਲ ਕਰਦਾ ਹੈ।

ਕਿਵੇਂ ਕੰਮ ਕਰਦੀ ਹੈ?

ਜਦੋਂ `try` ਬਲਾਕ ਵਿੱਚ ਕੋਈ ਐਕਸਪੈਸ਼ਨ ਹੁੰਦੀ ਹੈ, ਤਾਂ ਕੰਟਰੋਲ ਉਸ `catch` ਬਲਾਕ ਨੂੰ ਮਿਲਦਾ ਹੈ ਜੋ ਉਸ ਐਕਸਪੈਸ਼ਨ ਨੂੰ ਹੈਂਡਲ ਕਰਦਾ ਹੈ। ਜੇ ਕੋਈ ਮਿਲਦਾ ਜੁਲਦਾ `catch` ਬਲਾਕ ਨਹੀਂ ਮਿਲਦਾ, ਤਾਂ ਪ੍ਰੋਗਰਾਮ ਅਚਾਨਕ ਬੰਦ ਹੋ ਜਾਂਦਾ ਹੈ।

C++ ਦੇ ਮਿਆਰੀ ਐਕਸਪੈਸ਼ਨ

C++ ਨੇ `<exception>` ਹੈਡਰ ਵਿੱਚ ਮਿਆਰੀ ਐਕਸਪੈਸ਼ਨਾਂ ਦੀ ਇੱਕ ਵੰਸ਼ਾਵਲੀ ਦਿੱਤੀ ਹੈ, ਜਿਸਦਾ ਬੇਸ ਕਲਾਸ `std::exception` ਹੈ। ਕੁਝ ਆਮ ਮਿਆਰੀ ਐਕਸਪੈਸ਼ਨ ਹਨ:

- `std::runtime_error` — ਰਨਟਾਈਮ ਦੌਰਾਨ ਪਾਈਆਂ ਗਈਆਂ ਗਲਤੀਆਂ।
- `std::logic_error` — ਪ੍ਰੋਗਰਾਮ ਦੀ ਲਾਜਿਕ ਵਿੱਚ ਗਲਤੀਆਂ।
- `std::out_of_range` — ਗਲਤ ਅਤੇ ਜਾਂ ਕਨਟੇਨਰ ਇੰਡੈਕਸ ਤੱਕ ਪਹੁੰਚ।
- `std::invalid_argument` — ਗਲਤ ਫੰਕਸ਼ਨ ਆਰਗੂਮੈਂਟ।
- `std::bad_alloc` — ਮੈਮੋਰੀ ਐਲੋਕੇਸ਼ਨ ਫੇਲ੍ਹ।

ਇਹ ਮਿਆਰੀ ਐਕਸਪੈਸ਼ਨ ਡਿਵੈਲਪਰਾਂ ਨੂੰ ਆਮ ਗਲਤੀਆਂ ਨੂੰ ਚੁੱਢਣ ਅਤੇ ਸੰਭਾਲਣ ਵਿੱਚ ਮਦਦ ਕਰਦੇ ਹਨ, ਜਿਸ ਨਾਲ ਬਿਹਤਰ ਐਰਰ ਮੈਨੇਜਮੈਂਟ ਹੁੰਦਾ ਹੈ।

ਨਤੀਜਾ:

C++ ਸੰਗਠਿਤ ਐਕਸਪੈਸ਼ਨ ਹੈਂਡਲਿੰਗ ਨੂੰ ਸਮਰਥਨ ਦਿੰਦਾ ਹੈ, ਜੋ ਗਲਤੀਆਂ ਨੂੰ ਸੰਭਾਲ ਕੇ ਪ੍ਰੋਗਰਾਮ ਦੀ ਭਰੋਸੇਯੋਗਤਾ ਵਧਾਉਂਦਾ ਹੈ ਅਤੇ ਅਚਾਨਕ ਬੰਦ ਹੋਣ ਤੋਂ ਬਚਾਉਂਦਾ ਹੈ।

Q. What is the difference between an Object and a Class? What are the various access specifiers in C++?
Write a program to demonstrate working of different access specifiers. (Nov 22)

Ans. Difference Between Object and Class:

- A **Class** is a blueprint or template that defines the properties (data members) and behaviors (member functions) common to all objects of that type.
- An **Object** is an instance of a class. It represents a specific entity with actual values stored in memory.

Access Specifiers in C++:

C++ provides three access specifiers to control visibility of class members:

1. **Public:** Members are accessible from anywhere.
2. **Private:** Members are accessible only within the class.
3. **Protected:** Members are accessible within the class and by derived classes.

Program Demonstrating Access Specifiers:

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

class Demo {
public:
    int publicVar = 1;

private:
    int privateVar = 2;

protected:
    int protectedVar = 3;

public:
    void show() {
        cout << "Public: " << publicVar << endl;
        cout << "Private: " << privateVar << endl;
        cout << "Protected: " << protectedVar << endl;
    }
};

int main() {
    Demo obj;

    // Accessing public member directly
    cout << "Public: " << obj.publicVar << endl;

    // Accessing private or protected members directly causes error
    // cout << obj.privateVar; // Error
    // cout << obj.protectedVar; // Error

    obj.show(); // Access all members inside class method

    return 0;
}
```

This program shows that only public members are accessible outside the class, while private and protected members can only be accessed within class methods or derived classes.

ਵਸਤੂ (Object) ਅਤੇ ਕਲਾਸ (Class) ਵਿੱਚ ਅੰਤਰ:

ਕਲਾਸ (Class): ਕਲਾਸ ਇੱਕ ਨਕਸ਼ਾ ਜਾਂ ਟੈਂਪਲੇਟ ਹੁੰਦੀ ਹੈ ਜੋ ਕਿਸੇ ਕਿਸਮ ਦੀਆਂ ਸਾਰੀਆਂ ਵਸਤੂਆਂ (objects) ਲਈ ਆਮ ਗੁਣ (ਡਾਟਾ ਮੈਂਬਰ) ਅਤੇ ਵਿਹਾਰ (ਮੈਥਡ) ਨੂੰ ਪਰਿਭਾਸ਼ਿਤ ਕਰਦੀ ਹੈ।

ਵਸਤੂ (Object): ਵਸਤੂ ਕਲਾਸ ਦੀ ਇੱਕ ਨਮੂਨਾ (instance) ਹੁੰਦੀ ਹੈ। ਇਹ ਇੱਕ ਵਿਸ਼ੇਸ਼ ਏਨਟੀਟੀ ਹੈ ਜਿਸ ਵਿੱਚ ਅਸਲੀ ਕਦਰਾਂ ਮੈਮੋਰੀ ਵਿੱਚ ਸਟੋਰ ਹੁੰਦੀਆਂ ਹਨ।

C++ ਵਿੱਚ Access Specifiers (ਪਹੁੰਚ ਨਿਰਧਾਰਕ):

C++ ਤਿੰਨ ਪ੍ਰਕਾਰ ਦੇ access specifiers ਦਿੰਦਾ ਹੈ ਜੋ ਕਲਾਸ ਮੈਂਬਰਾਂ ਦੀ ਪਹੁੰਚ ਨੂੰ ਨਿਯੰਤਰਿਤ ਕਰਦੇ ਹਨ:

- **Public:** ਮੈਂਬਰਾਂ ਨੂੰ ਕਿਤੇ ਵੀ ਪਹੁੰਚ ਮਿਲ ਸਕਦੀ ਹੈ।
 - **Private:** ਮੈਂਬਰ ਸਿਰਫ਼ ਕਲਾਸ ਦੇ ਅੰਦਰ ਹੀ ਪਹੁੰਚਯੋਗ ਹੁੰਦੇ ਹਨ।
 - **Protected:** ਮੈਂਬਰ ਕਲਾਸ ਅਤੇ ਉਸ ਦੀਆਂ derived ਕਲਾਸਾਂ ਵਿੱਚ ਪਹੁੰਚਯੋਗ ਹੁੰਦੇ ਹਨ।
-

Access Specifiers ਦਾ ਪ੍ਰੋਗਰਾਮ:

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

class Demo {
public:
    int publicVar = 1;

private:
    int privateVar = 2;

protected:
    int protectedVar = 3;

public:
    void show() {
        cout << "Public: " << publicVar << endl;
        cout << "Private: " << privateVar << endl;
        cout << "Protected: " << protectedVar << endl;
    }
};

int main() {
    Demo obj;
    // Public ਮੈਂਬਰ ਨੂੰ ਸਿੱਧਾ ਐਕਸੈੱਸ ਕਰਨਾ
    cout << "Public: " << obj.publicVar << endl;

    // Private ਜਾਂ Protected ਮੈਂਬਰਾਂ ਨੂੰ ਸਿੱਧਾ ਐਕਸੈੱਸ ਕਰਨ 'ਤੇ Error ਆਵੇਗਾ
    // cout << obj.privateVar; // Error
    // cout << obj.protectedVar; // Error

    obj.show(); // ਕਲਾਸ ਦੇ ਮੈਥਡ ਵਿੱਚ ਸਾਰੇ ਮੈਂਬਰਾਂ ਨੂੰ ਐਕਸੈੱਸ ਕਰਨਾ
    return 0;
}
```

ਨਤੀਜਾ: ਇਸ ਪ੍ਰੋਗਰਾਮ ਵਿੱਚ ਦਿਖਾਇਆ ਗਿਆ ਹੈ ਕਿ ਸਿਰਫ਼ public ਮੈਂਬਰਾਂ ਨੂੰ ਕਲਾਸ ਤੋਂ ਬਾਹਰ ਸਿੱਧਾ ਐਕਸੈੱਸ ਕੀਤਾ ਜਾ ਸਕਦਾ ਹੈ, ਜਦਕਿ private ਅਤੇ protected ਮੈਂਬਰਾਂ ਨੂੰ ਸਿਰਫ਼ ਕਲਾਸ ਦੇ ਅੰਦਰ ਜਾਂ derived ਕਲਾਸਾਂ ਵਿੱਚ ਹੀ ਪਹੁੰਚ ਮਿਲਦੀ ਹੈ।