Class - 08

C++ class - 01

Class Note

C++ Foundations –> Functions, Inline Functions, Default Arguments, and Scope

1. The line `#include<iostream>` is used to include the input/output stream library which allows the use of `cin` and `cout` for taking input and displaying output. The line `using namespace std;` allows access to the standard library without writing `std::` each time.
2. `int var = 12;` declares a **global variable**. Global variables are accessible to all functions in the program unless a local variable with the same name hides them.
3. The function

```
inline int mx(int a, int b)
{
    return (a > b)? a : b ;
}
```

uses the **inline keyword**, suggesting to the compiler to replace the function call with its code body. This avoids the overhead of calling the function separately, making execution faster.

The ternary operator `(a > b) ? a : b` is used for a short conditional check—if `a` is greater than `b`, it returns `a`, otherwise it returns `b`. Inline functions are mainly used for small functions.

4. The function

```
inline int div(int a)
{
    return (!(a%2));
}
```

checks whether a number is even or odd. The expression `a % 2` gives remainder 0 for even numbers and 1 for odd numbers. Applying `!` (logical NOT) converts 0 to 1 (true) and 1 to 0 (false). So, it returns 1 for even numbers and 0 for odd numbers.

5. Inline functions are meant to **reduce function call overhead** and **speed up program execution**. However, if the inline function contains loops, recursion, or switch statements, the compiler may ignore the inline request and treat it as a normal function.
6. The function

```
void fun(int x, int y = var)
{
    int l;
    cout<<"x = "<<x<<", y = "<<y<<endl;
    var = 1;
    cout<<var<<endl;
}
```

demonstrates the concept of **default arguments**. The parameter `y` is given a default value equal to the global variable `var`. When the function is called with one argument, the second argument automatically takes the value of `var`.

7. In default arguments, the **rules** are:
   i) Default arguments must appear from right to left. If a parameter on the left has a default value, all parameters on its right must also have default values.
   ii) A **local variable** or another **parameter** cannot be used as a default value. Only **constants** or **global variables** are allowed as default argument values.
8. The line `void f(int z)` defines a simple function to show **local variable behavior**. Inside it, `z` is printed and modified, showing that parameter values are passed **by value**, meaning the original value in the caller function does not change.
9. Inside `main()`, the local variable `int v = 2;` hides the global variable `var` if both have the same name. The **scope resolution operator (::)** can be used to access the global version, e.g., `::var`.

10. The `cout << var << endl;` statement displays the current value of the global variable, which is `12` initially.
11. The comment `>>` and `<<` means the **extraction** and **insertion** operators.

    In `cin >> v >> f;`, data is extracted from the input stream, and in `cout << v;`, data is inserted into the output stream.

12. The `switch(v)` statement is used to demonstrate **multiple decision control**. It compares the value of `v` with each case. If a match is found, the corresponding block is executed. Example from the code:

```
switch(v)
{
    case 1:
        cout<<"the value of v is 1"<<endl;
        break;
    case 5:
        cout<<v<<endl;
        break;
    case 90:
        if(v > 0)
            cout<<"ok"<<endl;
        else
            cout<<"right"<<endl;
        break;
    default:
        //cout<<"default case executed"<<endl;
}
```

If none of the cases match, the `default` block executes .The `break` keyword stops the control from falling into the next case.

```
//fun(12);
//fun(12,89);
```

show that when `fun(12);` is called, `y` takes the default global variable `var = 12`, and when `fun(12,89);` is called, both values are explicitly passed.