2.8 FUNCTION PROTOTYPING [W-18]

- One of the most important features of C++ is the function prototype.
- A function prototype is a declaration that delivers the return type and the parameters of a function.
- A function is declared with a prototype.

- The function prototype tells the compiler the name of the function, the type of data returned by the function, the number of parameters the function expects to receive, the types of the parameters and the order in which these parameters are expected.
- The compiler uses function prototype to validate function calls.
- The function prototype is also called the function declaration.
- Functions are declared similar to variables, but they enclose their arguments in parenthesis [( )].
- The function prototype describes the function interface to the compiler by giving details like the numbers and type of arguments and the type of return values.
- **Syntax:** `return_type function_name (list of parameters);`
- **For examples:**

```
1. int add(x, y); //Declaration of add with x, y arguments.
2. int max( );    //Declaration of max with no argument.
```

2.9 INTERACTION BETWEEN FUNCTIONS

- The function [main() and other] interacts with each other through parameters. We pass the parameter to the function for the communication of calling function and the called function.
- There are two types of parameters actual parameters (used in the function call) and formal parameters (used in function definition).

2.9.1 Call by Reference [S - 18]

- When a function is called the address of the actual parameters are copied on to the formal parameters, though they may be referred by different variable names.
- This content of the variables that are altered within the function block are return to the calling function program. As the formal and the actual arguments are referencing the same memory location or address. This is known as call by reference, when we use this concept is functions.
  - Called function makes the alias of the actual variable which is passed.
  - Actual variables reflect the changes made on the alias of actual variables.
  - Number of values can be modified together.
  - For call by reference concept C++ uses a reference operator (&)
  - & operator allows true call by reference functions, eliminating the need to dereference arguments passed to the function.
  - The & operator is placed before the variable name of argument in the parameter list of the function. Then the function can be called without passing the address and without dereferencing within the function.

Program 2.12: Program to swap the values by call by reference.

```
#include<iostream.h>
void swapr (int &p, int &q)
{
    int dummy;
    dummy = p;
    p = q;
    q = dummy;
}
```

```
int main()
{
    int x = 225, y = 305;
    cout<<x<<"\t"<<y<<endl;
    swapr(x,y);
    cout<<"After swap"<<endl;
    cout<<x<<"\t"<<y;
}
```

Output:

```
225     305
After swap
305     225
```

2.9.2 Call by Value

- In call by value, we called function makes the copy of actual variable passed.
- The actual variables do not reflect the changes made on the copies of actual variables.
- In call by value only one value can be returned.

Program 2.13: Program for call by value.

```
#include<iostream.h>
void swap (int &p, int &q)
{
    int dummy;
    dummy = p;
    p = q;
    q = dummy;
}
int main()
{
    int x = 225, y = 305;
    cout<<x<<"\t"<<y<<endl;
    swap(x,y);
    cout<<"After swap"<<endl;
    cout<<x<<"\t"<<y;
}
```

Output:

```
225     305
After swap
305     225
```

Explanation of Program 4.6:

of type integer p, q and dummy. Hence, we

2.9.3 Return by Reference

[S - 19]

- The function can also return the reference of variable. This reference variable is actually an alias for the referred variable.
- The method of returning reference is used generally in operator overloading. This method will form a cascade of member function calls.

For example: `cout << k << a;`

- This statement is a set of cascaded calls which returns reference to the object cout.

the larger value. And this assigns the value - 1 to it.

2.10 INLINE FUNCTIONS [S - 18, 19; W - 18]

- C++ inline function is powerful concept that is commonly used with classes. If a function is inline, the compiler places a copy of the code of that function at each point where the function is called at compile time.
- Any change to an inline function could require all clients of the function to be recompiled because compiler would need to replace all the code once again otherwise it will continue with old functionality.
- The advantage of using function is to save the memory space. Consider a situation where a small function is called number of times calling a function requires certain overheads such as:
  - Passing to the function,
  - Passing the values or push arguments into the stack,
  - Save the registers,
  - Return back to calling function.

- In such a situation (if small program called number of times), execution time requires more. For the above problem, C programming language provides macros. But macros are called at execution time, so usual error checking does not occur during compilation.
- For the above situation, C++ compiler put the code directly inside the function body of calling program. Every time when function is called, at each place the actual code from the function would be inserted, instead of a jump to the function. Such functions are called inline functions.
- A function can be defined as an inline functions by writing the keyword inline to the function header as shown below:

```
inline function_name (list of arguments)
{
    //function body
}
```

- When we do not specify keyword inline then only one copy of function code (object code) is called depending upon the functions calls. This is illustrated in Fig. 2.6.

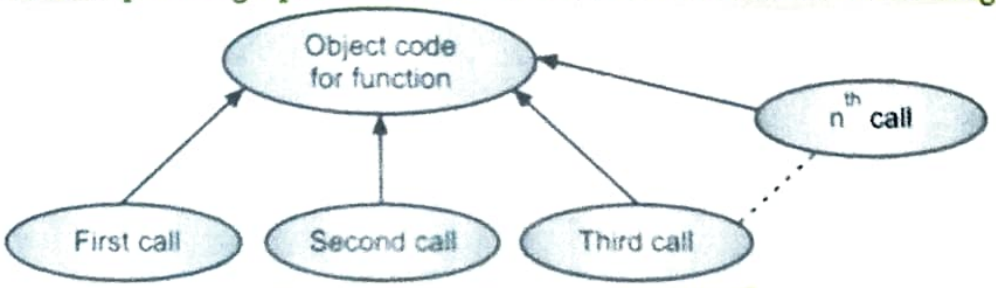

Fig. 2.6: Calls Given to a Function

- But as we said above, lot of time is needed for all these calls to execute. If we make this function inline then the calling statement is replaced with the function code. So it minimizes the time spent on jumping, pushing arguments and so on. This is shown in Fig. 2.7.

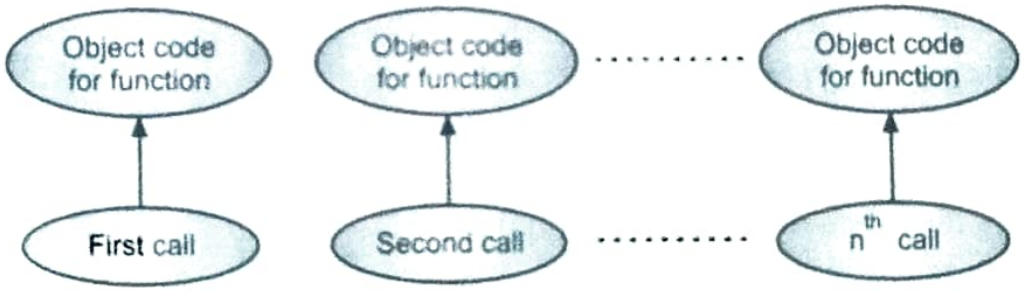

Fig. 2.7: Calls given to Inline Function

Program 2.16: Program illustrates the use of inline function to compute square of a number. [S - 18]

```
#include<iostream.h>
inline int sq (int x)
{
    int p;
    p = x * x;
    return p;
}
```

```
int main()
{
    int n;
    cout<<"Enter number";
    cin>>n;
    cout<<"square is"<<sq(n)<<endl;
}
```

Output:

```
Enter number 12
square is 144
```

2.11 DEFAULT ARGUMENTS