Default Arguments (or) Default parameter values

* Default arguments are the default values provided for the function parameters, if the arguments are not passed during function call.
  - the default values are assigned automatically by the compiler.

(Eg)
```
# include <iostream.h>

int sum (int x, int y , int z=10)
 {
   return x+y+z ;
 }
void main ()
 {
  cout << sum (20, 30) << endl; // Prints 60 => one missing argument
  cout << sum( 20, 30,50) <<endl; // Prints 100 => No missing argument.
 }
```
Note:

Defaults should be added from right to left.

(Eg)
```
int sum (int x , int y=20 , int z) ; // illegal
int sum ( int x, int y=20 , int z=10) ; // legal.
```

Reference Variable

* A reference variable is an alternative name (alias) for already existing variable.

* Once a reference is initialized with a variable, either the variable name or reference name may be used to refer the variable.

Syntax :

$$\boxed{\texttt{datatype\& referencename = variablename;}}$$

(Eg)

```
int a = 10;
int& b = a ; // b is an alternative name to a. Both
                                       represent same data in memory
cout << a << " " << b ; // both prints 10 .
b = b+20;
cout << a << " " << b ; // both prints 30
```

Note :

A reference variable must be initialized at the time of declaration.

(Eg)

```
int x[10];
int& y = x[10];
```

Reference variables with Functions

* In C++, a function can take arguments passed by
  - value → a copy of actual parameters in function call is assigned to formal parameters.
  - pointer → address of actual parameters is passed.
  - reference → an alias (reference) of actual parameters is passed

(Eg) Program to swap integer values using reference variable (call by reference)

```
#include <iostream.h>
void swap (int& x, int& y)
 {
  int t;
  t=x; x=y; y=t;
 }
void main()
 {
 int a,b;
 cout << "Enter two integers\n";
 cin >> a >> b;
 swap (a,b);
 cout << "After swapping" << a << " " << b;
 }
```

* Here formal arguments x and y in the called function become aliases to actual arguments a and b in calling function.
* Both represent same data in memory.

Call by reference Using pointers.

```
#include <iostream.h>
void swap (int *x, int *y)
{
 int t;
 t = *x; *x = *y; *y=t;
}
void main()
 {
 int a, b;
  :
 swap (&a, &b);
  :
 }
```

Note:
* In C, call by reference is achieved by pointers and indirection operator.
* It is also allowed in C++, but more complicated than reference variables.

Inline Functions

* An inline function is a function that is expanded in line when it is invoked.
  (ie) The compiler replaces the function call with the corresponding function code during compilation.

Syntax:

```
inline fn.header          (Eg) inline int cube(int a)
{                                {
  fn body                          return (a* a* a);
}                                }

                                 c = cube (3);
```

* All inline functions must be defined before they are called.
* Prefix the keyword "inline" to the function definition.
* Usually, functions are made inline, when they are small enough to be defined in one or two lines.
* Inline fns should not be recursive

Why inline fn?

* For smaller programs, the time needed to make function call is more than the execution time of function.
* Hence inline functions are used to reduce function call overhead.

Function call execution steps:

* CPU stores memory address of instruction following fn call
* Copies fn args. to stack.
* Transfer control to specified fn.
* Execute fn. code.
* Stores return value in a register.
* Returns to calling fn.