

Call by reference

The functions, which you saw earlier in the various examples were capable of returning only one result in the calling environment. Moreover, in all the examples actual parameter and formal parameter had different memory locations. And so any changes done on formal parameters in the function were not reflected back in the actual parameter such parameters are known as Value parameters. Now, we will be doing Call by Reference, which will allow using the same memory location for Actual and Formal parameter. It will help us to get the changes of Formal parameters reflected in the Actual parameters. Some important points about Call by reference in C++ are as follows:

1. In Formal parameter, the reference parameters are prefixed with & (ampersand) symbol.
2. In Actual parameter, we are not allowed to use constant value and expression for sending values to reference parameter.

<p>Example 11 a:</p> <pre>void Change(int A,int &B) { //A is Value, B is Reference parameter A+=B; B+=A; cout<<A<<"&"<<B<<endl; } void main() { int P=100,Q=200; Change(P,Q); //Function Call 1 cout<<P<<"#"<<Q<<endl; Change(500,P); //Function Call 2 cout<<P<<"#"<<Q<<endl; Change(P+Q,P); //Function Call 3 cout<<P<<"#"<<Q<<endl; }</pre> <p>Note: Kindly note, actual parameters in Function Calls 1, 2 and 3 - we are using Variable, Value and Expression in the first parameter. The same is valid as while calling a function with Value parameter, we can send the value from Variable/Constant/Expression. The same is not valid while sending the value to Reference Parameter. It means, we can't write Constant/Expression in Actual parameter to be sent to reference parameter.</p> <p>Output: 300&500 100#500 600&700 700#500 1900&2600 2600#500</p>	<p>Example 11 b (Same as 11 a with Function Prototype):</p> <pre>void Change(int ,int &); //Function Prototype void main() { int P=100,Q=200; Change(P,Q); //Function Call 1 cout<<P<<"#"<<Q<<endl; Change(500,P); //Function Call 2 cout<<P<<"#"<<Q<<endl; Change(P+Q,P); //Function Call 3 cout<<P<<"#"<<Q<<endl; } void Change(int A,int &B) { A+=B; B+=A; cout<<A<<"&"<<B<<endl; }</pre> <p>Tabular Representation of Changes in the variables used in 11 a and 11 b.</p> <table border="1"><thead><tr><th>P</th><th>Q</th><th>A</th><th>&B</th></tr></thead><tbody><tr><td>100</td><td>200</td><td>100</td><td>200</td></tr><tr><td></td><td>500</td><td>300</td><td>500</td></tr><tr><td></td><td></td><td>500</td><td>100</td></tr><tr><td>700</td><td></td><td>600</td><td>700</td></tr><tr><td></td><td></td><td>1200</td><td>700</td></tr><tr><td>2600</td><td></td><td>1900</td><td>2600</td></tr></tbody></table>	P	Q	A	&B	100	200	100	200		500	300	500			500	100	700		600	700			1200	700	2600		1900	2600
P	Q	A	&B																										
100	200	100	200																										
	500	300	500																										
		500	100																										
700		600	700																										
		1200	700																										
2600		1900	2600																										
<p>Example 12: Program with a function to calculate Income Tax and Allowance for the Employees.</p> <pre>void Calc(float BS,float &IT, float &ALL) { IT=BS*0.1; //Income Tax 10% of Basic Salary ALL=BS*0.6; //Allowance 60% of Basic Salary } void main() { float B,I,A; cout<<"Enter Basic Salary:"<<cin>>B; Calc(B,I,A); }</pre>	<p>Example 13: Program with a function to exchange (swap) the content of two variables.</p> <pre>void Swap(int &A,int &B) { //A and B is Reference parameter T=A; A=B; B=T; } void main() { int P,Q; cout<<"First Number:"<<cin>>P;</pre>																												

```
cout<<"Income Tax:"<<I<<endl;
cout<<"Allowance      : "<<A<<endl;
cout<<"Salary in hand:"<<B+A-I<<endl;
}
```

```
cout<<"Second Number:"<<cin>>Q;
if (P<Q)
    Swap(P,Q);
cout<<"Big Number:"<<P<<endl;
cout<<"Small Number:"<<Q<<endl;
}
```

Default Parameter: A formal parameter, which carries a default value to be substituted in absence of corresponding actual parameter is known as a default parameter.

Example:

```
void Over(int Balls=6);
```

Default parameter has the following characteristics:

1. Default parameters can not be referenced.
i.e. `void Over(int &Balls=6);` will be wrong.
2. When default and non-default parameters are together, the default parameters are required to be after non-defaults.
i.e. `void Over(int Runs, int Balls=6);` is correct and
`void Over(int Balls=6, int Runs);` is incorrect
3. When the program has function prototype for the function with default parameter, the default value is only substituted in the function prototype and not in the function header.
i.e.

```
float Over(int Runs, int Balls=6);
void main()
{
    int R,B;
    cout<<"Enter Runs & Balls:"<<cin>>R>>B;
    cout<<"Average          : "<<Over(R,B)<<endl;
    cout<<"Enter Runs        : "<<cin>>R;
    cout<<"Average          : "<<Over(R)<<endl;
}
float Over(int Runs, int Balls)
{
    return Runs/Balls;
}
```
4. When we have more than one default parameter, the values of actual parameters are substituted to the corresponding formal parameters i.e. from beginning only.

```
void Line(int N=5, char CH='*')
```

```
{
    for (int C=1; C<=N; C++)
        cout<<CH;
    cout<<endl;
}
```

```
void main()
```

```
{
    Line(8);      //N will get 8 CH will remain *
    Line();       //N and CH both will get default values
    Line(4, '#'); //N will get 4 and CH will be #
}
```

Output:

```
*****
*****
####
```

How to give line gap in this

Global Variable: A variable, which is declared in the beginning outside all the functions is known as a global variable. A global variable is accessible in all the functions and can be modified in any function.

Local Variable: A variable, which is declared within a function or a compound statement (within { }) is known as a local variable. A local variable is accessible and modified only within its scope.

```
#include <iostream.h>
const int MAX=100;//Global Constant
int A=10,B=20;    //Global Variables A and B
void Modify(int &K)
{
    int T=A+B;      //Local Variable T within function Modify function
    K+=(MAX-T);
    A+=10;
    B+=5;
    cout<<A<<B<<T<<endl;
}
void main()
{
    int A=5;        //Local Variable A within function main function
    Modify(A);
    cout<<A<<": "<<B<<endl;
    cout<<::A<<endl;//We need to use scope resolution operator :: for
                    //referring to global variable, if a local variable in
                    //the same module has the same name.
}
//Like in this example the global/local variables A
```

Output:

```
20:25:30
75:25
20
```

Practice Questions:

1. Find output of the following (Assuming the programs contain all required header files):

(a)	(b)
<pre>void DOIT(int &A,int B=10) { A+=B; B+=A; cout<<A<<"&"<<B<<endl; } void main() { int P=100,Q=200; DOIT(P,Q); cout<<P<<": "<<Q<<endl; DOIT(Q); cout<<P<<": "<<Q<<endl; }</pre>	<pre>void DISP(int A=3,int B=3) { for (int I=1;I<=B;I++) cout<<A; cout<<endl; } void main() { int M=5,N=5; DISP(); DISP(M,N); DISP(N); }</pre>

<p>(c)</p> <pre> int N=100; void Compute(int &A) { A++; N+=A; cout<<A<<"#"<<N<<endl; } void main() { int N=100; Compute(N); cout<<N<<"@"<<: :N<<endl; Compute(N); cout<<: :N<<"@"<<N<<endl; } </pre>	<p>(d)</p> <pre> int G=10; void Calculate(int &M,int N=1) { M+=N; cout<<M<<"#"<<G++<<endl; } void main() { int P=50,Q=100; Calculate(P); cout<<P<<": "<<Q<<": "<<G<<endl; Calculate(P); cout<<P<<": "<<Q<<": "<<G<<endl; } </pre>
--	---

2. Debug the following (Assuming the programs contain all required header files):

<p>(a)</p> <pre> void Assign(int A=10,int B) { A++; B++; cout<<A<<"&"<<B<<endl; } void main() { int P=100,Q=200; Assign(P=5); cout<<P<<": "<<Q<<endl; Assign(Q,P); cout<<P<<": "<<Q<<endl; } </pre>	<p>(b)</p> <pre> void DISP(int &A=3) { if A==3 cout<<"I am default"<<endl; else cout<<"I am not"<<endl; } void main() { int M=50; DISP(M,3); DISP(); } </pre>
---	---

Answers (To be checked only after attempting the questions):

- | | |
|---|--|
| 1. (a) 300&500
300:200
210&220
300:210 | (b) 333
55555
555 |
| (c) 101#201
101@201
102#303
303@102 | (d) 51#10
51:100:11
52#11
52:100:12 |
2. (a) void Assign(int A,int B=10)
Assign(P);
- (b) void DISP(int A=3)
if (A==3)
DISP(M);