Problem Solving and Programing

Bitwise Operators in C/C++

In C, following 6 operators are bitwise operators (work at bit-level)

- **& (bitwise AND)** Takes two numbers as operands and does AND on every bit of two numbers. The result of AND is 1 only if both bits are 1.

- **| (bitwise OR)** Takes two numbers as operands and does OR on every bit of two numbers. The result of OR is 1 any of the two bits is 1.

- **^ (bitwise XOR)** Takes two numbers as operands and does XOR on every bit of two numbers. The result of XOR is 1 if the two bits are different.

- **<< (left shift)** Takes two numbers, left shifts the bits of the first operand, the second operand decides the number of places to shift.

- **>> (right shift)** Takes two numbers, right shifts the bits of the first operand, the second operand decides the number of places to shift.

- **~ (bitwise NOT)** Takes one number and inverts all bits of it

```c
/* C Program to demonstrate use of bitwise operators */
#include<stdio.h>
int main()
{
    unsigned char a = 5, b = 9; // a = 5(00000101), b = 9(00001001)
    printf("a = %d, b = %d\n", a, b);
    printf("a&b = %d\n", a&b); // The result is 00000001
    printf("a|b = %d\n", a|b);  // The result is 00001101
    printf("a^b = %d\n", a^b); // The result is 00001100
    printf("~a = %d\n", a = ~a);   // The result is 11111010
    printf("b<<1 = %d\n", b<<1);  // The result is 00010010
    printf("b>>1 = %d\n", b>>1);  // The result is 00000100
    return 0;
}
```

```
a = 5, b = 9
a&b = 1
 a|b = 13
a^b = 12
~a = 250
b = 4
```

Functions

Introduction to Functions

- A complex problem is often easier to solve by dividing it into several smaller parts, each of which can be solved by itself.
- This is called structured programming.
  - Split a large problem into smaller pieces.

  - Why ?
    - Easy to understand. Easy to code.
    - Re–use of code (Functions can be called several times in the same program, allowing the code to be reused - avoids code repetition)
      - How it is different from iterations (loops) ?
      - Iterations are used when the same code is repeated at the same place again and again.
      - By using Functions the same code can be used at different parts of the program .

Modules – Communication (Functions Communication)

- Issues in the process are:
  - How do modules communicate – transfer values from one module to another?
  - What is the flow of control from one module to another?
  - Is there a need for one module to know the details of another module?
  - Can they share some common information?
  - What happens to a module after execution?
  - What happens to the names and other information in a modules?

Advantages of Functions

- Functions separate the concept (what is done) from the implementation (how it is done).
- Functions make programs easier to understand.
- Functions can be called several times in the same program, allowing the code to be reused.

C++ Functions

- In C++ we use **functions** also referred to as **modules** to perform specific tasks that we have identified in our solution

- C++ allows the use of both internal (user-defined) and external functions.

- External functions (e.g., **abs**, **ceil**, **rand**, sqrt, etc.) are usually grouped into specialized libraries (e.g., **iostream, stdlib**, **math**, etc.)

- C++ Programs combine user-defined functions with library functions.

User-Defined Functions

- C++ programs usually have the following form:

```
// include statements
// function prototypes
// main() function
// function definitions
```

Function Input and Output

Function Prototype

- Every function should have a function prototype.

- Like a variable declaration the function prototype tells the compiler
    1. The name of the function
    2. The type of arguments (not necessary the name of the arguments)
    3. The type of return type
    4. The function will be defined later

return-data-type function-name(argument data types);

or

void function-name(argument data types);

Function Prototype (cont.)

- The use of function prototypes permits *error checking* of data types by the compiler.
- It also ensures conversion of all arguments passed to the function to the declared argument data type when the function is called.

Function definition

- A function definition specifies
  1. The name of the function
  2. The types and number of parameters it expects to receive
  3. Its return type.
  4. Function body with the declarations of its local variables, and the statements to perform the specific task assigned to the function.

Function definition

Return data type

Name of the function

Arguments (or parameter list)

```
type function_name (type , type )
{

    statements;
}
```

Statements

- Variable declaration
- Operations
- Return value (if any)

Function Definition

```
return-data-type function-name(parameter list)
{
 constant declarations
 variable declarations

 other C++ statements

 return value
}
```

For example: Definition of a function that computes the absolute value of an integer:

```
int absolute(int x){
    if (x >= 0)  return x;
    else         return -x;
}
```

Function Definition (cont.)

- Non value-returning functions

```
void function-name(parameter list)
{
 constant declarations
 variable declarations

 other C++ statements
}
```

Function Definition (cont.)

- The argument names in the function header are referred to as *formal parameters*.

```
int FindMax(int x, int y)
{
 int maximum;

 if(x>=y)
   maximum = x;
 else
   maximum = y;

 return maximum;
}
```

Calling a function

- A function is *called* by specifying its name followed by its arguments.
- Non-value returning functions:

  function-name (data passed to function);
- Value returning functions:

  results = function-name (data passed to function);

Calling a function (cont.)

```
#include <iostream.h>

int FindMax(int, int); // function prototype

int main()
{
 int firstnum, secnum, max;

 cout << "\nEnter two numbers: ";
 cin >> firstnum >> secnum;

 max=FindMax(firstnum, secnum); // the function is called here

 cout << "The maximum is " << max << endl;

 return 0;
}
```

```
int FindMax(int x, int y)
{
 int maximum;

 if(x>=y)
   maximum = x;
 else
   maximum = y;

 return maximum;
}
```

- The argument names in the function call are referred to as *actual parameters*

```
#include<iostream>
using namespace std;
int findSmallest (int[],int);
int main ()
{
    int myarray[10] = {11,5,2,20,42,53,23,34,101,22};
    int pos,temp,pass=0;
    cout<<"\n Input list of elements to be Sorted\n";
    for(int i=0;i<10;i++)
    {
        cout<<myarray[i]<<"\t";
    }
    for(int i=0;i<10;i++)
    {
        pos = findSmallest (myarray,i);
        temp = myarray[i];
        myarray[i]=myarray[pos];
        myarray[pos] = temp;
        pass++;
    }
    cout<<"\n Sorted list of elements is\n";
    for(int i=0;i<10;i++)
    {
        cout<<myarray[i]<<"\t";
    }
    cout<<"\nNumber of passes required to sort the array: "<<pass;
    return 0;
}
```

```
int findSmallest(int myarray[],int i)
{
    int ele_small,position,j;
    ele_small = myarray[i];
    position = i;
    for(j=i+1;j<10;j++)
    {
        if(myarray[j]<ele_small)
        {
            ele_small = myarray[j];
            position=j;
        }
    }
    return position;
}
```

C++ program to find the sum of diagonals of a matrix

```
#include<iostream>
 int main()
{
      int a[5][5],d1sum=0,d2sum=0,m,i,j;
      cout<<"Enter size of the square matrix(max 5):";
      cin>>m;
      cout<<"\nEnter the Matrix row wise:\n";
      for(i=0;i<m;i++)
      for(j=0;j<m;++j)
          cin>>a[i][j];
      for(i=0;i<m;++i)
      for(j=0;j<m;++j)
      {
            if(i==j)
            d1sum+=a[i][j];
            if(i+j==(m-1))
            d2sum+=a[i][j];
      }
      cout<<"\nSum of 1st diagonal is "<<d1sum;
      cout<<"\nSum of 2nd diagonal is "<<d2sum;
       return 0;
}
```

```
Enter size of the square matrix(max 5):

3

1 2 3

4 5 6

7 8 9

Sum of 1st diagonal is 15

Sum of 2nd diagonal is 15
```

C++ Program to Find Sum of Elements Above and Below Main Diagonal of Matrix

C++ Program to Print Upperhalf and Lowerhalf Triangle of Square Matrix

```
int main()
{
    int a[10][10],i,j,m;
    cout<<"Enter size of the Matrix(min:3,max:5):";
    cin>>m;
    cout<<"\nEnter the Matrix row wise:\n";
    for(i=0;i<m;i++)
    for(j=0;j<m;++j)
        cin>>a[i][j];
    cout<<"\n\n";
    for(i=0;i<m;++i)
    {
        for(j=0;j<m;++j)
        {
            if(i<j)
            cout<<a[i][j]<<" ";
            else
            cout<<"  ";
        }
        cout<<"\n";
    }
    cout<<"\n\n";
    for(i=0;i<m;++i)
    {
        for(j=0;j<m;++j)
        {
            if(j<i)
            cout<<a[i][j]<<" ";
            else
            cout<<" ";
        }
        cout<<"\n";
    }

    return 0;
}
```

```
Enter size of the Matrix(min:3,max:5):3

Enter the Matrix row wise:
1 2 3
4 5 6
7 8 9


2 3
  6

4
7 8
```

Calling a function by value

- The communication between calling function (or caller) and called function (or calle) done by sending arguments to called function.

- The calling function sends the arguments in two ways –
  - call by value –
  - call by reference (For now, we focus on call by value).

- Call by value
  - Copy of argument passed to function
  - Changes in function do not effect original
  - Use when function does not need to modify argument
    - Avoids accidental changes

Function parameters

- Function parameters
  - Actual parameter
    - Actual parameters (also known as *arguments*) are what are passed by the caller.
  - Formal parameters
    - Formal parameters are the parameters as they are known in the function definition.
  - Formal parameters must match with actual parameters in *order*, *number* and *data type*.
  - If the type is not the same, type conversion will be applied (coercion of arguments). But this might cause some errors (Ex - double to int) so you need to be careful!

Example – Factorial of a number

```
#include <iostream>
using namespace std;
int fact(int);        // prototype of user defined function fact
int main()
{
   int t = 5, s;      //  function fact is called
   s = fact(t);       // main is the calling function and fact is the called f
   cout << " The factorial of 5 is : " << s;
   return 0;
}
```

t	5
s	

Example – Factorial of a number

```
#include <iostream>
using namespace std;
int fact(int);        // prototype of user defined function fact
int main()
{
   int t = 5, s;       //  function fact is called
   s = fact(t);        // main is the calling function and fact is the called fu
   cout << " The factorial of 5 is : " << s;
   return 0;
}
int fact (int n)
{ int f = 1, i = 1;   // local variables of function fact
    while( i < = n) { f = f * i; i++; }
     return f;
 }
```

t	5
s	

n	5
f	1
i	1

Example – Factorial of a number

```
#include <iostream>
using namespace std;
int fact(int);        // prototype of user defined function fact
int main()
{
   int t = 5, s;      //  function fact is called
   s = fact(t);       // main is the calling function and fact is the called f
   cout << " The factorial of 5 is : " << s;
   return 0;
}
int fact (int n)
{ int i = 1, f = 1;   // local variables of function fact
    while( i < = n) { f = f * i; i++; }
     return f;
 }
```

t	5
s	

n	5
i	2
f	2

Example – Factorial of a number

```
#include <iostream>
using namespace std;
int fact(int);        // prototype of user defined function fact
int main()
{
   int t = 5, s;      //  function fact is called
   s = fact(t);       // main is the calling function and fact is the called f
   cout << " The factorial of 5 is : " << s;
   return 0;
}
int fact (int n)
{ int i = 1, f = 1;   // local variables of function fact
    while( i < = n) { f = f * i; i++; }
     return f;
 }
```

t	5
s	

n	5
i	3
f	6

Example – Factorial of a number

```
#include <iostream>
using namespace std;
int fact(int);        // prototype of user defined function fact
int main()
{
   int t = 5, s;      //  function fact is called
   s = fact(t);       // main is the calling function and fact is the called f
   cout << " The factorial of 5 is : " << s;
   return 0;
}
int fact (int n)
{ int i = 1, f = 1;   // local variables of function fact
    while( i < = n) { f = f * i; i++; }
     return f;
 }
```

t	5
s	

n	5
i	4
f	24

Example – Factorial of a number

```
#include <iostream>
using namespace std;
int fact(int);        // prototype of user defined function fact
int main()
{
   int t = 5, s;      //  function fact is called
   s = fact(t);       // main is the calling function and fact is the called f
   cout << " The factorial of 5 is : " << s;
   return 0;
}
int fact (int n)
{ int i = 1, f = 1;   // local variables of function fact
    while( i < = n) { f = f * i; i++; }
     return f;
 }
```

t	5
s	

n	5
i	5
f	120

Example – Factorial of a number

```
#include <iostream>
using namespace std;
int fact(int);        // prototype of user defined function fact
int main()
{
   int t = 5, s;       //  function fact is called
   s = fact(t);       // main is the calling function and fact is the called f
   cout << " The factorial of 5 is : " << s;
   return 0;
}
int fact (int n)
{ int i = 1, f = 1;   // local variables of function fact
    while( i < = n) { f = f * i; i++; }
     return f;     // it returns 120
 }
```

t	5
s	

n	5
i	6
f	120

Example – Factorial of a number

```
#include <iostream>
using namespace std;
int fact(int);        // prototype of user defined function fact
int main()
{
   int t = 5, s;      //  function fact is called
   s = fact(t);       // main is the calling function and fact is the called f
   cout << " The factorial of 5 is : " << s;
   return 0;
}
```

t	5
s	120

Functions - Terminology

- function return
  - Return statement terminates execution of the current function
  - Control returns to the calling function (the function who calls this function)
  - if **return** **is an expression then**
    - The value of **expression** is returned as the value of the function call
    - Only one value can be returned (Exception for arrays).
  - The return type is not mandatory, in case of no return type the return type must be void (keyword)- at the function prototype and also at the function definition.

Swapping of two numbers – Does it work?

```
#include <iostream>
using namespace std;
int swap(int, int);        // prototype of user defined function swap
int main()
{
   int a, b;
   cout >> "input the values to be exchanged:";
   cin >>   a >> b;
   cout << "Values of a and b before exchange : " << a << b << endl;
   swap (a, b);
   cout << "Values of a and b after exchange : " << a << b << endl;
   return 0;
}
int swap (int x, int y)
{
    int temp = x; x = y; y = temp;
    return 0;
}
```

Swapping of two numbers – Does it work?

```
#include <iostream>
using namespace std;
int swap(int, int);        // prototype of user defined function swap
int main()
{
   int a, b;
   cout >> "input the values to be exchanged:";
   cin >>   a >> b;
   cout << "Values of a and b before exchange : " << a << b << endl;
   swap (a, b);
   cout << "Values of a and b after exchange : " << a << b << endl;
   return 0;
}
int swap (int x, int y)
{
    int temp = x; x = y; y = temp;
    return 0;
}
```

Swapping of two numbers – Does it work?

```
#include <iostream>
using namespace std;
int swap(int, int);        // prototype of user defined function swap
int main()
{
   int a, b;
   cout >> "input the values to be exchanged:";
   cin >>   a >> b;
   cout << "Values of a and b before exchange : " << a << b << endl;
   swap (a, b);
   cout << "Values of a and b after exchange : " << a << b << endl;
   return 0;
}
```

1020	a	
2030	b	

Swapping of two numbers – Does it work?

```
#include <iostream>
using namespace std;
int swap(int, int);        // prototype of user defined function swap
int main()
{
   int a, b;
   cout >> "input the values to be exchanged:";
   cin >>   a >> b;
   // Assume that user enters the values for a and b as 7 and 9
   cout << "Values of a and b before exchange : " << a << b << endl;
   swap (a, b);
   cout << "Values of a and b after exchange : " << a << b << endl;
   return 0;
}
```

1020	**a**	**7**
2030	b	9

Swapping of two numbers – Does it work?

```
#include <iostream>
using namespace std;
int swap(int, int);        // prototype of user defined function swap
int main()
{
   int a, b;
   cout >> "input the values to be exchanged:";
   cin >>   a >> b;
   // Assume that user enters the values for a and b are 7 and 9
   cout << "Values of a and b before exchange : " << a << b << endl;
   swap (a, b);
   //swap (7, 9)
   cout << "Values of a and b after exchange : " << a << b << endl;
   return 0;
}
```

1020	**a**	**7**
2030	b	9

Swapping of two numbers – Does it work?

```
#include <iostream>
using namespace std;
int swap(int, int);        // prototype of user defined function swap
int main()
{
   int a, b;
   cout >> "input the values to be exchanged:";
   cin >>   a >> b;
   cout << "Values of a and b before exchange : " << a << b << endl;
   swap (a, b);
   cout << "Values of a and b after exchange : " << a << b << endl;
   return 0;
}
int swap (int x, int y)
{


}
```

1020	a	7
2030	b	9

3040	x	
3080	y	

Swapping of two numbers – Does it work?

```
#include <iostream>
using namespace std;
int swap(int, int);        // prototype of user defined function swap
int main()
{
   int a, b;
   cout >> "input the values to be exchanged:";
   cin >>   a >> b;
   cout << "Values of a and b before exchange : " << a << b << endl;
   swap (a, b);
   cout << "Values of a and b after exchange : " << a << b << endl;
   return 0;
}
int swap (int x, int y)
{
    int temp = x;

}
```

1020	a	7
2030	b	9

3040	x	7
3080	y	9
5010	temp	7

Example – swapping of two numbers

```
#include <iostream>
using namespace std;
int swap(int, int);        // prototype of user defined function swap
int main()
{
   int a, b;
   cout >> "input the values to be exchanged:";
   cin >>   a >> b;
   cout << "Values of a and b before exchange : " << a << b << endl;
   swap (a, b);
   cout << "Values of a and b after exchange : " << a << b << endl;
   return 0;
}
int swap (int x, int y)
{
    int temp = x;
               x = y;
}
```

1020	a	7
2030	b	9

3040	x	9
3080	y	9
5010	temp	7

Example – swapping of two numbers

```
#include <iostream>
using namespace std;
int swap(int, int);        // prototype of user defined function swap
int main()
{
   int a, b;
   cout >> "input the values to be exchanged:";
   cin >>   a >> b;
   cout << "Values of a and b before exchange : " << a << b << endl;
   swap (a, b);
   cout << "Values of a and b after exchange : " << a << b << endl;
   return 0;
}
int swap (int x, int y)
{
    int temp = x;
             x = y;
             y = temp;
}
```

1020	a	7
2030	b	9

3040	x	9
3080	y	7
5010	temp	7

Example – swapping of two numbers

```
#include <iostream>
using namespace std;
int swap(int, int);        // prototype of user defined function swap
int main()
{  int a, b;
   cout >> "input the values to be exchanged:";
   cin >>   a >> b;
   cout << "Values of a and b before exchange : " << a
   swap (a, b);
   cout << "Values of a and b after exchange : " << a << b << endl;
   return 0;
}
int swap (int x, int y)
{
    int temp = x;
              x = y;
             y = temp;
    return 0;
}
```

1020	a	7
2030	b	9

Swapping of two numbers – Does it work?

```
#include <iostream>
using namespace std;
int swap(int, int);        // prototype of user defined function swap
int main()
{
   int a, b;
   cout >> "input the values to be exchanged:";
   cin >>   a >> b;
   cout << "Values of a and b before exchange : " << a << b << endl;
   swap (a, b);
   cout << "Values of a and b after exchange : " << a << b << endl;
   return 0;
}
```

1020	a	7
2030	b	9

Output generated is
Input the values to be exchanged:
Values of a and b before exchange: 7 9
Values of a and b after exchange: 7 9

Swapping of two numbers – Alternate Approach

```
#include <iostream>
using namespace std;
void swap(int *, int *); //prototype of user defined function swap. why * ?
int main()
{  int a, b;
   cout >> "input the values to be exchanged:";
   cin >>   a >> b;
   cout << "Values of a and b before exchange: " << a << b;
   swap (&a, &b);  // addresses of  identifiers a and b
   cout << "\n Values of a and b after exchange : " << a << b;
   return 0;
}
// swap receives the addresses of actual arguments
void swap (int *x, int *y)
{
    int temp;
    temp =*x; // *x = the value in the memory location x
   *x  = *y;  // The value in memory location y is stored  in memory location x
   *y  = temp; // The value in memory location temp is stored  in memory location y
}
```

Swapping of two numbers – Alternate Approach

```
#include <iostream>
using namespace std;
void swap(int *, int *); //prototype of user defined function swap. why * ?
int main()
{ int a, b;
   cout >> "input the values to be exchanged:";
   cin >>   a >> b;
   cout << "Values of a and b before exchange: " << a << b;
   swap (&a, &b);  // addresses of  identifiers a and b
   cout << "\n Values of a and b after exchange : " << a << b;
   return 0;
}
// swap receives the addresses of actual arguments
void swap (int *x, int *y)
{



}
```

1060	a	
2090	b	

Swapping of two numbers – Alternate Approach

```
#include <iostream>
using namespace std;
void swap(int *, int *); //prototype of user defined function swap. why * ?
int main()
{ int a, b;
   cout >> "input the values to be exchanged:";
   cin >>   a >> b;
   cout << "Values of a and b before exchange: " << a << b;
   swap (&a, &b);  // addresses of identifiers a and b
   cout << "\n Values of a and b after exchange : " << a << b;
   return 0;
}
// swap receives the addresses of actual arguments
void swap (int *x, int *y)
{



}
```

1060	a	7
2090	b	9

Swapping of two numbers – Alternate Approach

```
#include <iostream>
using namespace std;
void swap(int *, int *); //prototype of user defined function swap. why * ?
int main()
{ int a, b;
   cout >> "input the values to be exchanged:";
   cin >>   a >> b;
   cout << "Values of a and b before exchange: " << a << b;
   swap (&a, &b);  // addresses of  identifiers a and b
   cout << "\n Values of a and b after exchange : " << a << b;
   return 0;
}
// swap receives the addresses of actual arguments
void swap (int *x, int *y)
{
    int temp;


}
```

1060	a	7
2090	b	9

3060	temp	
6060	X	1060
6070	Y	2090

Swapping of two numbers – Alternate Approach

```
#include <iostream>
using namespace std;
void swap(int *, int *); //prototype of user defined function swap. why * ?
int main()
{ int a, b;
   cout >> "input the values to be exchanged:";
   cin >>   a >> b;
   cout << "Values of a and b before exchange: " << a << b;
   swap (&a, &b);  // addresses of identifiers a and b
   cout << "\n Values of a and b after exchange : " << a << b;
   return 0;
}
// swap receives the addresses of actual arguments
void swap (int *x, int *y)
{
    int temp;
    temp = *x; // *x = the value in the memory location x

}
```

1060	a	7
2090	b	9

3060	temp	7
6060	X	1060
6070	Y	2090

Swapping of two numbers – Alternate Approach

```
#include <iostream>
using namespace std;
void swap(int *, int *); //prototype of user defined function swap. why * ?
int main()
{ int a, b;
  cout >> "input the values to be exchanged:";
  cin >>  a >> b;
  cout << "Values of a and b before exchange: " << a << b;
  swap (&a, &b);  // addresses of identifiers a and b
  cout << "\n Values of a and b after exchange : " << a << b;
  return 0;
}
// swap receives the addresses of actual arguments
void swap (int *x, int *y)
{
    int temp;
    temp =*x; // *x = the value in the memory location x
   *x  = *y;  // The value in memory location y is stored  in memory location x

}
```

1060	a	9
2090	b	9

3060	temp	7
6060	X	1060
6070	Y	2090

Swapping of two numbers – Alternate Approach

```
#include <iostream>
using namespace std;
void swap(int *, int *); //prototype of user defined function swap. why * ?
int main()
{ int a, b;
  cout >> "input the values to be exchanged:";
  cin >>  a >> b;
  cout << "Values of a and b before exchange: " << a << b;
  swap (&a, &b);  // The arguments are the addresses of  variables
  cout << "\n Values of a and b after exchange : " << a << b;
  return 0;
}
// swap receives the addresses of actual arguments
void swap (int *x, int *y)
{
    int temp;
    temp =*x; // *x = the value in the memory location x
   *x  = *y;  // The value in memory location y is stored  in memory location x
   *y  = temp; // // The value in memory location temp is stored  in memory location y
}
```

1060	a	9
2090	b	7

3060	temp	7
6060	X	1060
6070	Y	2090

Swapping of two numbers – Alternate Approach

```
#include <iostream>
using namespace std;
void swap(int *, int *); //prototype of user defined function swap. why * ?
int main()
{ int a, b;
   cout >> "input the values to be exchanged:";
   cin >>   a >> b;
   cout << "Values of a and b before exchange: " << a << b;
   swap (&a, &b);  // The arguments are the addresses of  variables
   cout << "\n Values of a and b after exchange : " << a << b;
   return 0;
}
```

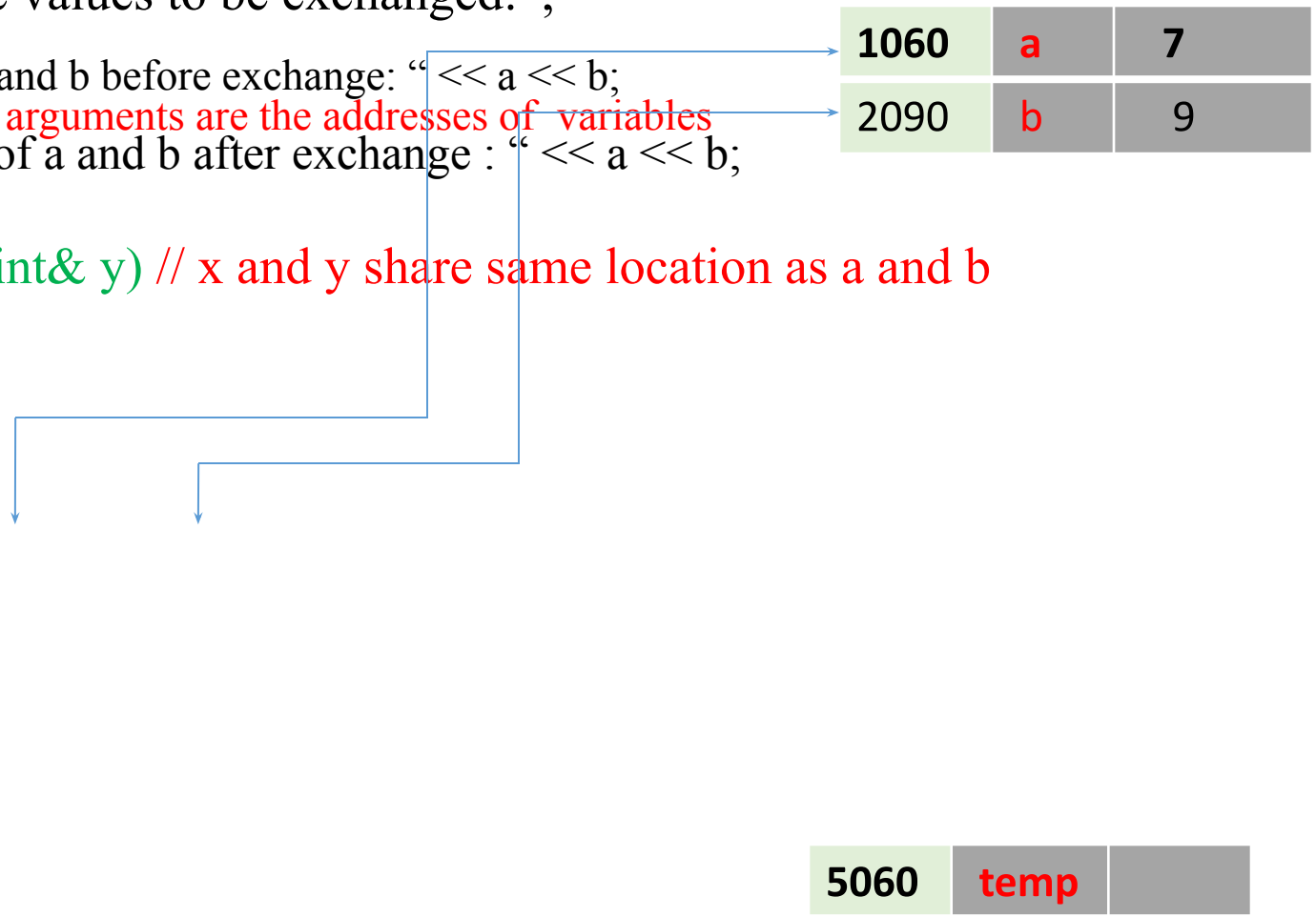
1060	a	9
2090	b	7

Swapping of two numbers – Call by reference

```
#include <iostream>
using namespace std;
void swap(int&, int&);
int main()
{ int a, b;
   cout >> "input the values to be exchanged:";
   cin >>   a >> b;
   cout << "Values of a and b before exchange: " << a << b;
   swap (a, b);  // The arguments are the addresses of  variables
   cout << "\n Values of a and b after exchange : " << a << b;
   return 0;
}
void swap (int& x, int& y) // x and y share same location as a and b
{
     int temp;
     temp = x;
     x  = y
     y  = temp;
}
```

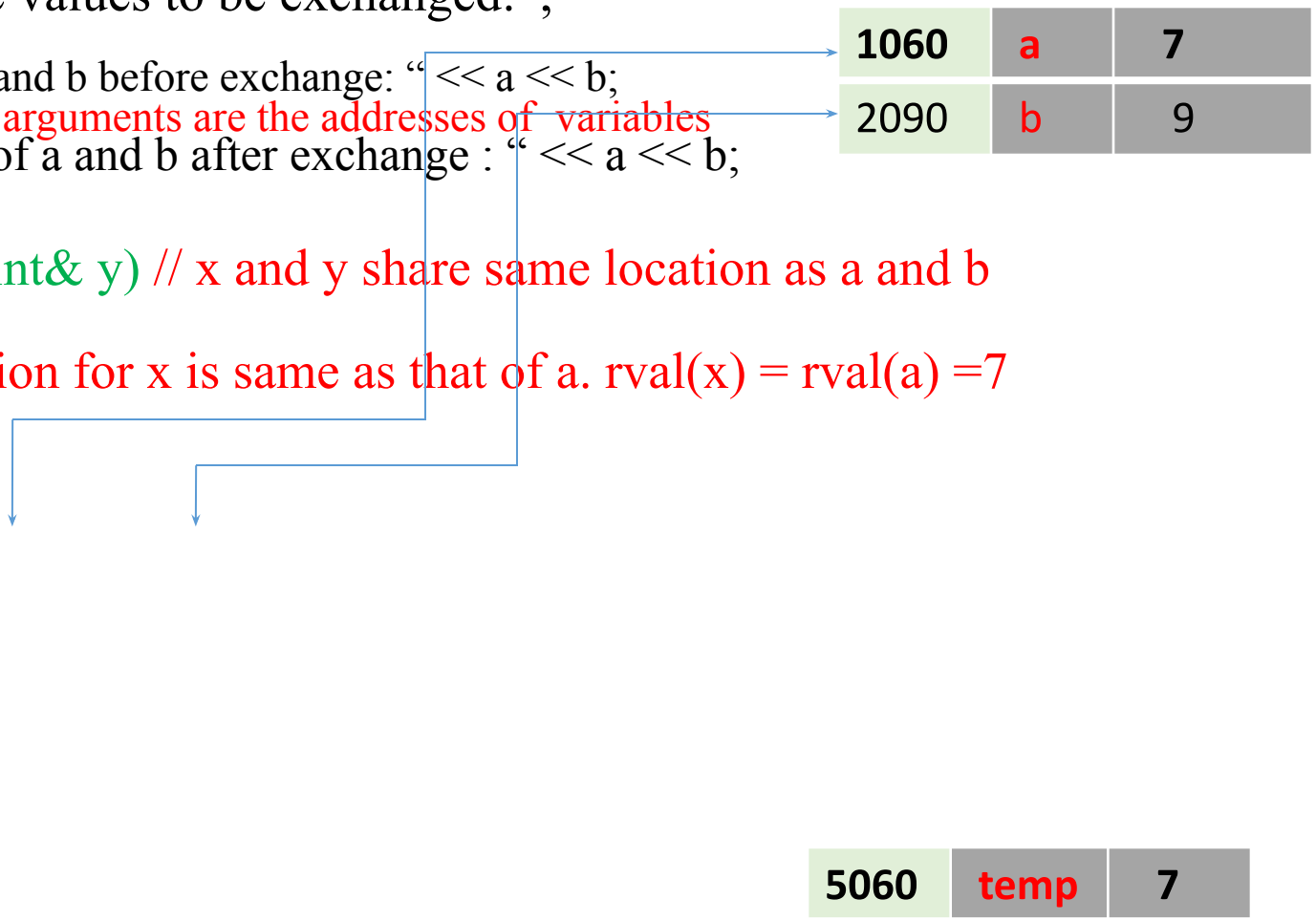
Swapping of two numbers – Call by reference

```
#include <iostream>
using namespace std;
void swap(int&, int&);
int main()
{ int a, b;
   cout >> "input the values to be exchanged:";
   cin >>   a >> b;
   cout << "Values of a and b before exchange: " << a << b;
   swap (a, b);  // The arguments are the addresses of  variables
   cout << "\n Values of a and b after exchange : " << a << b;
   return 0;
}
void swap (int& x, int& y) // x and y share same location as a and b
{
    int temp;

}
```



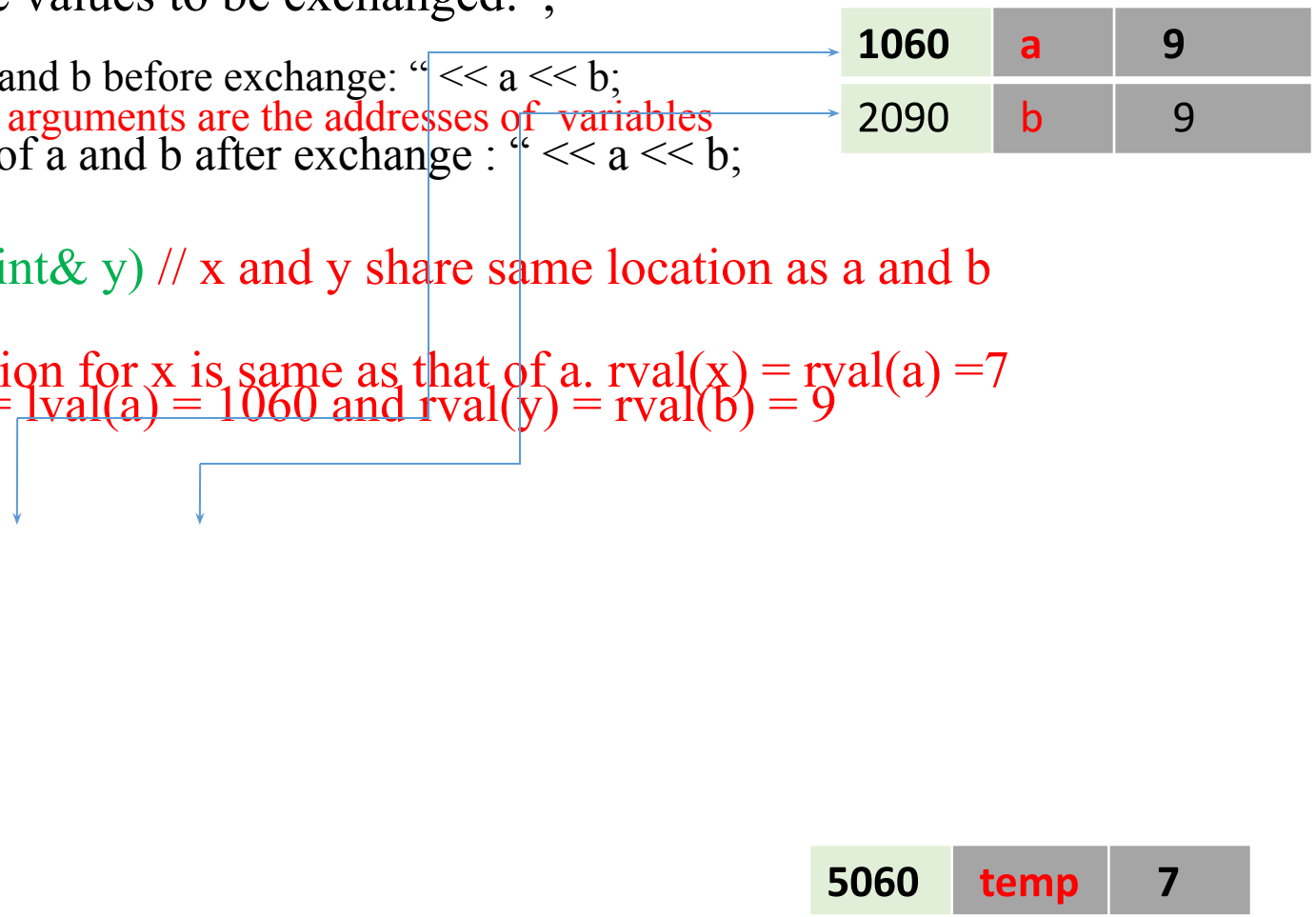
Swapping of two numbers – Call by reference

```
#include <iostream>
using namespace std;
void swap(int&, int&);
int main()
{ int a, b;
   cout >> "input the values to be exchanged:";
   cin >>   a >> b;
   cout << "Values of a and b before exchange: " << a << b;
   swap (a, b);  // The arguments are the addresses of  variables
   cout << "\n Values of a and b after exchange : " << a << b;
   return 0;
}
void swap (int& x, int& y) // x and y share same location as a and b
{
    int temp;
    temp = x; //location for x is same as that of a. rval(x) = rval(a) =7

}
```



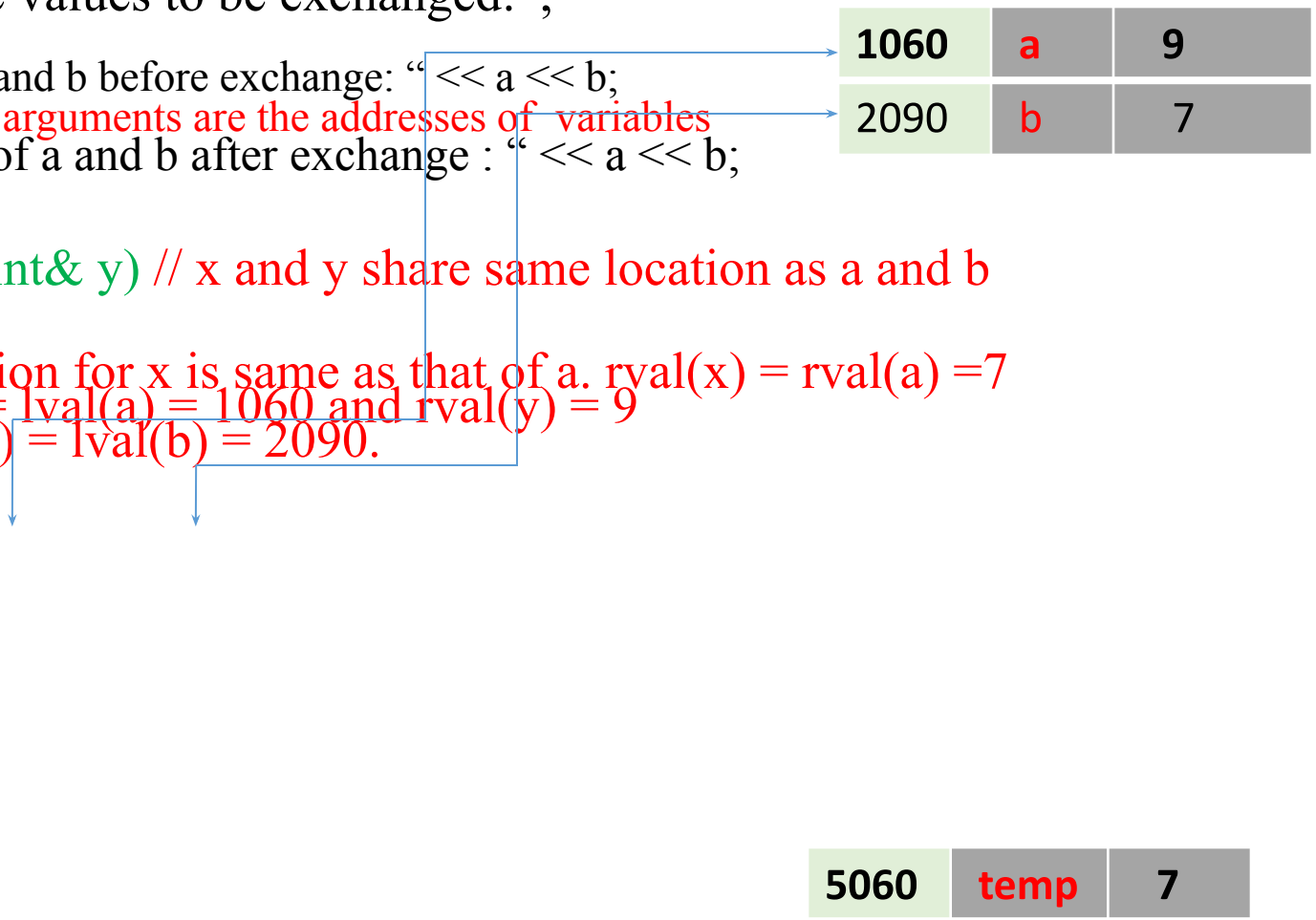
Swapping of two numbers – Call by reference

```
#include <iostream>
using namespace std;
void swap(int&, int&);
int main()
{ int a, b;
   cout >> "input the values to be exchanged:";
   cin >>   a >> b;
   cout << "Values of a and b before exchange: " << a << b;
   swap (a, b);  // The arguments are the addresses of  variables
   cout << "\n Values of a and b after exchange : " << a << b;
   return 0;
}
void swap (int& x, int& y) // x and y share same location as a and b
{
    int temp;
    temp = x; //location for x is same as that of a. rval(x) = rval(a) =7
    x  = y // lval(x) = lval(a) = 1060 and rval(y) = rval(b) = 9
}
```



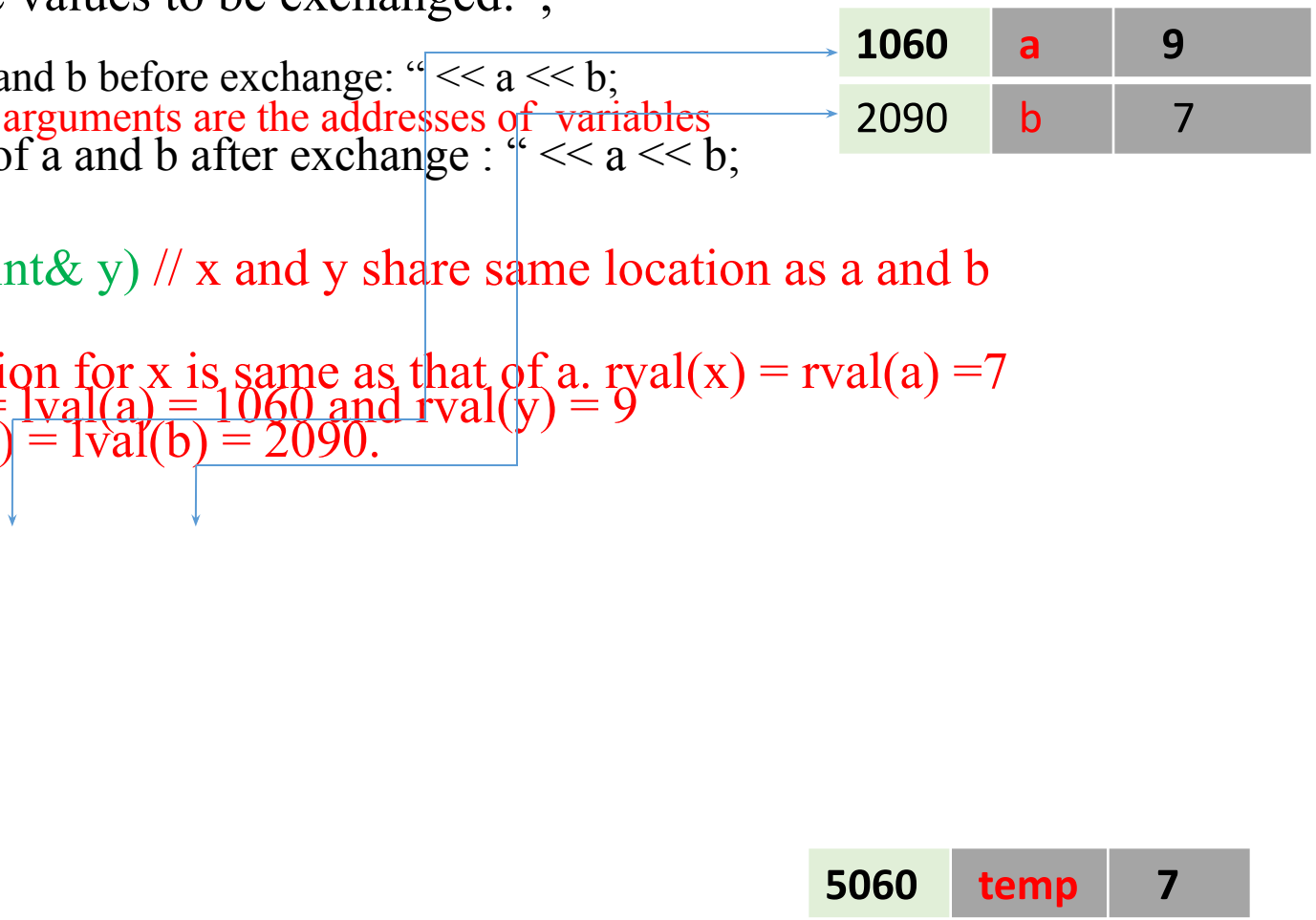
Swapping of two numbers – Call by reference

```
#include <iostream>
using namespace std;
void swap(int&, int&);
int main()
{ int a, b;
   cout >> "input the values to be exchanged:";
   cin >>   a >> b;
   cout << "Values of a and b before exchange: " << a << b;
   swap (a, b);  // The arguments are the addresses of  variables
   cout << "\n Values of a and b after exchange : " << a << b;
   return 0;
}
void swap (int& x, int& y) // x and y share same location as a and b
{
     int temp;
     temp = x; //location for x is same as that of a. rval(x) = rval(a) =7
     x  = y // lval(x) = lval(a) = 1060 and rval(y) = 9
     y  = temp; lval(y) = lval(b) = 2090.
}
```



Swapping of two numbers – Call by reference

```
#include <iostream>
using namespace std;
void swap(int&, int&);
int main()
{ int a, b;
   cout >> "input the values to be exchanged:";
   cin >>   a >> b;
   cout << "Values of a and b before exchange: " << a << b;
   swap (a, b);  // The arguments are the addresses of  variables
   cout << "\n Values of a and b after exchange : " << a << b;
   return 0;
}
void swap (int& x, int& y) // x and y share same location as a and b
{
     int temp;
     temp = x; //location for x is same as that of a. rval(x) = rval(a) =7
     x  = y // lval(x) = lval(a) = 1060 and rval(y) = 9
     y  = temp; lval(y) = lval(b) = 2090.
}
```



Swapping of two numbers – Call by reference

```
#include <iostream>
using namespace std;
void swap(int&, int&);
int main()
{ int a, b;
   cout >> "input the values to be exchanged:";
   cin >>   a >> b;
   cout << "Values of a and b before exchange: " << a << b;
   swap (a, b);  // The arguments are the addresses of  variables
   cout << "\n Values of a and b after exchange : " << a << b;
   return 0;
}
```

1060	**a**	**9**
2090	b	7

Very Simple Exercises

1. If a four-digit number is input through the keyboard, write a program to obtain the sum of the first and last digit of this number.
2. In a town, the percentage of men is 52. The percentage of total literacy is 48. If total percentage of literate men is 35 of the total population, write a program to find the total number of illiterate men and women if the population of the town is 80,000.
3. If the total selling price of 15 items and the total profit earned on them is input through the keyboard, write a program to find the cost price of one item.
4. If a five-digit number is input through the keyboard, write a program to print a new number by adding one to each of its digits. For example if the number that is input is 12391 then the output should be displayed as 23402.
5. If cost price and selling price of an item is input through the keyboard, write a program to determine whether the seller has made profit or incurred loss. Also determine how much profit he made or loss he incurred.

Very Simple Exercises

6. A five-digit number is entered through the keyboard. Write a program to obtain the reversed number and to determine whether the original and reversed numbers are equal or not.
7. Write a program to check whether a triangle is valid or not, when the three angles of the triangle are entered through the keyboard. A triangle is valid if the sum of all the three angles is equal to 180 degrees.
8. Given the length and breadth of a rectangle, write a program to find whether the area of the rectangle is greater than its perimeter. For example, the area of the rectangle with length =5 and breadth = 4 is greater than its perimeter.
9. Given three points (x1, y1), (x2, y2) and (x3, y3), write a program to check if all the three points fall on one straight line.
10. Given the coordinates (x, y) of a center of a circle and its radius, write a program which will determine whether a given point lies inside the circle, on the circle or outside the circle.

Printing Calendar for a given Year

- Develop functional decomposition and write a C++ program to print the calendar of an year, given the day of the week of January first of the year.
- Assumptions:
  - Input is year, an integer > 1700
  - Input the day of Jan first, an integer 0 to 6 for Sunday through Saturday.

Printing Calendar for a given Year

- Functional Decomposition:
  - Input and validate year and day of day of Jan 1.
  - To check whether given year is leap.
  - To find number of days of month for each month.
  - Day of first of every month.
  - To Print header for the calendar
  - To Print month name and week header
  - To Print calendar for the month

Printing Calendar for a given Year

- If the year is 2012 and Jan 1, 2012 is 0 (Sun), Output should be

```
                2012
             January
 S   M   T   W   T   F   S
 1   2   3   4   5   6   7
      ........
            February
 S   M   T   W   T   F   S
                 1   2   3   4
 5   6   7   8   9  10  11
12  13  14  15  16  17  18
19  20  21  22  23  24  25
26  27  28  29
```

Compare ages of two persons

- DOB of two persons are given as date, month, and year. Validate inputs and determine elder person
- One possible solution:
  - Compare years, then months and then days
- Is there an alternate solution?
  - Compute an integer for each person based on the date of birth.
  - Compare the two integers.

Average

- In a class there are n number of students – each studying m subjects. Marks of each of the student in each of the subjects are to be read and class average in each subject and the average of total marks in all subjects to be computed.
- There is no need to store the values – No need to use arrays – Assume that m is 3 and use sentinel -100 to end inputs. Output marks in each subject and total obtained by each student in a row and subject average and class average in the last row.

Some more functions

- Write a function to take a floating point number as input and returns the same number rounded to k decimal places. Do not use any system defined functions. If input is 17.24578, and k = 2, the output is 17.25 and 345.2034 is rounded as 345.20.
- Use functions to write a program to implement integer calculator with functions add, subtract, multiply, quotient, modulus, exponent. Provide proper user interface.

Some more functions

- Write a menu driven program that allows a user to enter five numbers and then choose between finding the smallest, largest, sum, average or median. The menu and all are to be functions. Use switch statement to determine the action to be taken.
- Write a program to read principal amount, rate of interest and time in years and calculate simple interest and compound interest compounded every k months.

Rules for using Default Arguments

- When we mention a default value for a parameter while declaring the function, it is said to be as default argument.

- Only the last argument must be given default value. You cannot have a default argument followed by non-default argument.

```
sum (int x,int y);
sum (int x,int y=0);
sum (int x=0,int y); // This is Incorrect
```

- If you default an argument, then you will have to default all the subsequent arguments after that.

```
sum (int x,int y=0);
sum (int x,int y=0,int z); // This is incorrect
sum (int x,int y=10,int z=10); // Correct
```

- You can give any value a default value to argument, compatible with its datatype.

```
#include <iostream>
int main()
{
	if (2)
		std::cout << "hello";
	else
		std::cout << "world";
	return 0;
}
```

hello

```
#include <iostream>
int main()
{
	if (NULL)
		std::cout << "hello";
	else
		std::cout << "world";
	return 0;
}
```

```
#include <iostream>
int main()
{
	if (0)
		std::cout << "hello";
	else
		std::cout << "world";
	return 0;
}
```

world

NULL is an equivalent of 0 world

What is the output of following program?

```
#include <iostream>
using namespace std;
int main()
{
        int i, j, k;
        int sum[2][4];
        for (i = 0; i < 2; i++)
                {
                for (j = 0; j < 3; j++)
                        sum[i][j];
        }
        cout << sum[i][j];
        return 0;
}
```

In this program, in given array we did not assign any value so it will be pointing to any garbage value.

```
#include <iostream>
using namespace std;
int main()
{
        int i, x[5], y, z[5];
        for (i = 0; i < 5; i++) {
                x[i] = i;
                z[i] = i + 3;
                y = z[i];
                x[i] = y++;
        }
        for (i = 0; i < 5; i++)
                cout << x[i] << " ";
        return 0;
}
```

3 4 5 6 7

```
#include <iostream>
using namespace std;
int max(int& x, int& y, int& z)
{
	if (x > y && y > z) {
		y++;
		z++;
		return x++;
	} else {
		if (y > x)
			return y++;
		else
			return z++;
	}
}
int main()
{
	int A, B;
	int a = 10, b = 13, c = 8;
	A = max(a, b, c);
	cout << a++ << " " << b-- << " " << ++c << endl;
	B = max(a, b, c);
	cout << ++A << " " << --B << " " << c++ << endl;
	return 0;
}
```

```
#include <iostream>
using namespace std;
void fun(int& a, int b)
{
	a += 2;
	b += 1;
}
int main()
{
	int x = 10, y = 2;
	fun(x, y);
	cout << x << " " << y << " ";
	fun(x, y);
	cout << x << " " << y;
	return 0;
}
```

12 2 14 2

```
#include <iostream>
using namespace std;
void f2(int p = 30)
{
	for (int i = 20; i <= p; i += 5)
		cout << i << " ";
}
void f1(int& m)
{
	m += 10;
	f2(m);
}
int main()
{
	int n = 20;
	f1(n);
	cout << n << " ";
	return 0;
}
```

20 25 30 30

```
#include <iostream>
using namespace std;
int main()
{
	int x[] = { 12, 25, 30, 55, 110 };
	int* p = x;
	while (*p < 110) {
		if (*p % 3 != 0)
			*p = *p + 1;
		else
			*p = *p + 2;
		p++;
	}
	for (int i = 4; i >= 1; i--) {
		cout << x[i] << " ";
	}
	return 0;
}
```

110 56 32 26

```
#include <iostream>
using namespace std;
int main()
{
	char* str = "GEEKSFORGEEK";
	int* p, arr[] = { 10, 15, 70, 19 };
	p = arr;
	str++;
	p++;
	cout << *p << " " << str << endl;
	return 0;
}
```

15 EEKSFORGEEK

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

int main()
{
	int a[] = { 1, 2 }, *p = a;
	cout << p[1];
}
```

We can access array using ‘p’ just like with ‘a’

2

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

int main()
{
        int a[] = { 10, 20, 30 };
        cout << *a + 1;
}
```

*a refers to 10 and adding a 1 to it gives 11.

```
#include <iostream>
using namespace std;
    void foo();
    int main()
    {
        void foo();
        foo();
        return 0;
    }
    void foo()
    {
        cout<<“2 “;
    }
```

2