GeeksMan

We code therefore
we are

CODE FOUNDATION

Functions

A function is a block of code which only runs when it is called. On being called it performs a set of tasks.
Till now you are familiar with the main function only. Now we will study how to make functions and it’s uses. Functions are important for reusing code: Define the code once, and use it many times.
A **function declaration** tells the compiler about a function's name, return type, and parameters. A **function definition** provides the actual body of the function.

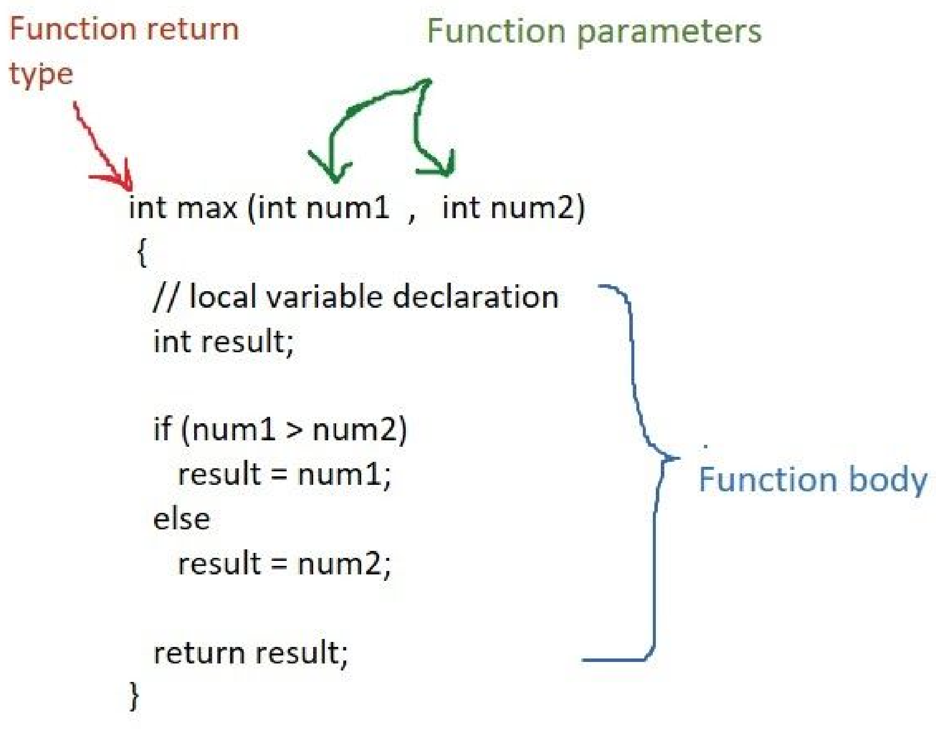
Form of C++ function:

```
return_type function_name( parameter list ) {

  body of the function

}
```

TERMS

- **Return Type** − **A function may return a value. The return_type is the data type of the value the function returns. Some functions perform the desired operations without returning a value. In this case, the return_type is the keyword void.**
- **Function Name** − **This is the actual name of the function. The function name and the parameter list together constitute the function signature.**
- **Parameters** − **A parameter is like a placeholder. When a function is invoked, you pass a value to the parameter. This value is referred to as an actual parameter or argument. The parameter list refers to the type, order, and number of the parameters of a function. Parameters are optional; that is, a function may contain no parameters.**
- **Function Body** − **The function body contains a collection of statements that define what the function does.**

Example of a function to calculate the maximum of 2 numbers:



```
int max (int num1 ,  int num2)
{
  // local variable declaration
  int result;

  if (num1 > num2)
    result = num1;
  else
    result = num2;

  return result;
}
```

Function declaration:

int max(int num1, int num2);
We need to provide information about the function to the compiler so function declaration includes it's return type, name and its arguments. Note that it's not necessary to write the arguments name. We can only write the datatype of arguments
Like this: **int max(int, int);**

NOTE: Number of parameter and their data type while calling must match with function signature.

Calling a Function

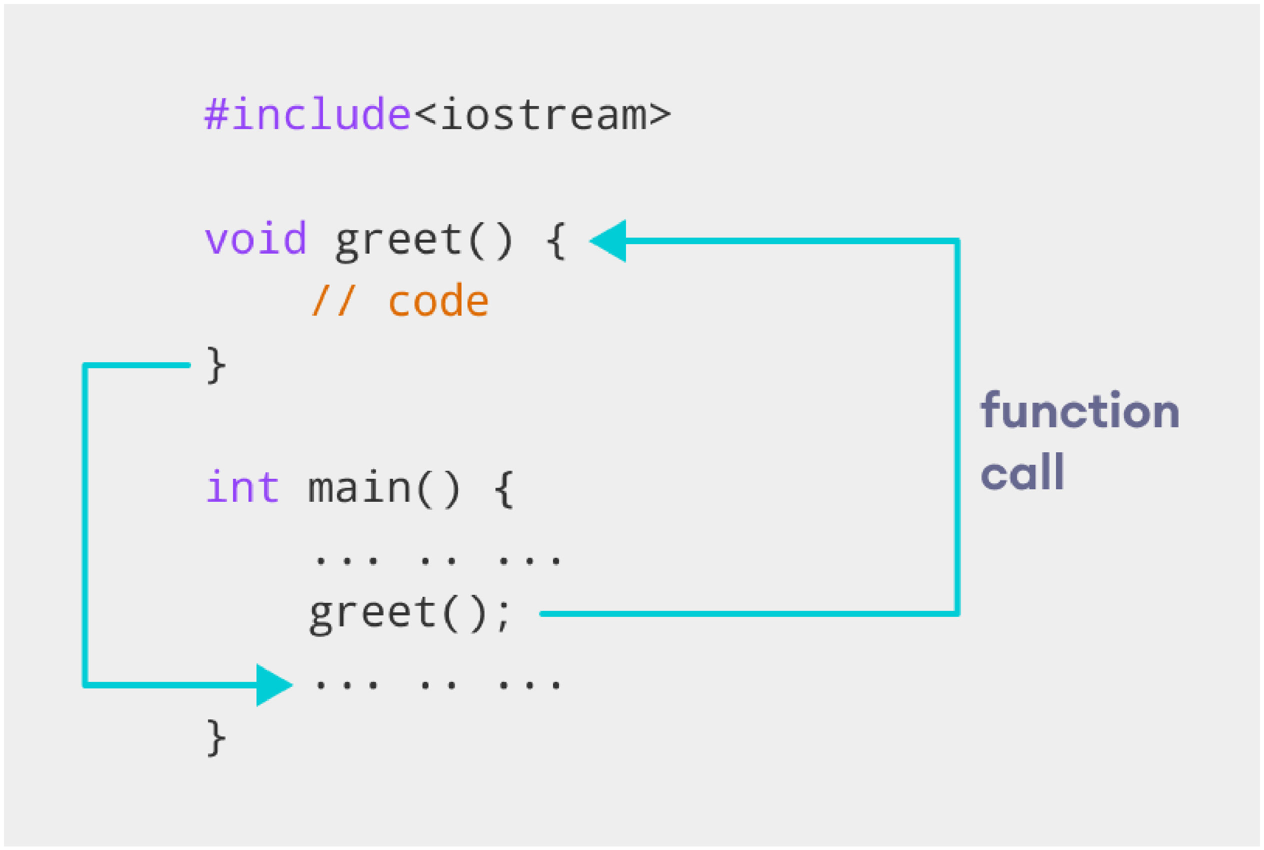
Here's an example of a function declaration. // function declaration

```
void greet() {
   cout << "Hello World";
}
```

In the above program, we have declared a function named greet(). To use the greet() function, we need to call it.

Here's how we can call the above greet() function.

```
int main() {

   // calling a function
   greet();

}
```



```
#include<iostream>

void greet() {
    // code
}

int main() {
    ... .. ...
    greet();
    ... .. ...
}
```

When a program calls a function, program control is transferred to the called function. A called function performs a defined task and when it's return statement is executed or when its function-ending closing brace is reached, it returns program control back to the main program.

To call a function, you simply need to pass the required parameters along with the function name, and if the function returns a value, then you can store the returned value.

This is an example of a function which finds the maximum of 2 numbers.

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

// function declaration
int max(int num1, int num2);

int main () {
  // local variable declaration:
  int a = 100;
  int b = 200;
  int ret;

  // calling a function to get max value.
  ret = max(a, b);
  cout << "Max value is : " << ret << endl;

  return 0;
}
int max(int num1, int num2) {
   // local variable declaration
   int result;

   if (num1 > num2)
      result = num1;
   else
      result = num2;

   return result;
}

```

In this example we are storing the result returned by function max() into a variable ret.It is absolutely okay if we don’t store the variable and just print the returned value directly like **cout<<max(2,3)<<endl;**

Function Prototype

In C++, the code of function declaration should be before the function call. However, if we want to define a function after the function call, we need to use the function prototype. For example,

```
// function prototype
void add(int, int);

int main() {
   // calling the function before declaration.
   add(5, 3);
   return 0;
}

// function definition
void add(int a, int b) {
   cout << (a + b);
}
```

In the above code, the function prototype is:

```
void add(int, int);
```

This provides the compiler with information about the function name and its parameters. That's why we can use the code to call a function before the function has been defined.

The syntax of a function prototype is:

```
returnType functionName(dataType1, dataType2, ...);
```

Example : C++ Function Prototype

```
// using function definition after main() function
// function prototype is declared before main()
```

```
#include <iostream>

using namespace std;

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

int main() {
    int sum;

    // calling the function and storing
    // the returned value in sum
    sum = add(100, 78);

    cout << "100 + 78 = " << sum << endl;

    return 0;
}

// function definition
int add(int a, int b) {
    return (a + b);
}
```

Output

```
100 + 78 = 178
```

Quick questions:

Question 1:

You will be given two numbers(both integers) and you need to return their sum. For this problem, what should be the return type of function?

We need to return the sum of two integers, which is again an integer. So the sum that we want to return is of type "int". Hence return type should be "int" for this function.

Question 2:

If a function only prints "Hello World" and doesn't return anything, what will be it's return type?

void return type is used for functions which don't return anything.

Example:

```
void demo(int a, int b){
   cout<<"Hello World"; }
```

Question 3

What will be the output?

```
void demo(int x, int y){

   cout << x << " " << y;

}

int main() {

   int a = 5;

   int b = 15;

   demo(b);

}
```

This code snippet will give a compilation error. In the function call we are passing only one argument but there are two arguments in the function declaration.That's why there will be compilation error.

Need of functions:

Why are we using functions?What's their need? Let's look for their benefits.

- **Neat and clear code:** Code inside a function is neat and easy to dry run. If you want to do the same thing again and again, just make it's function and invoke it wherever you need it.
- **Reusable code:** As already explained the repetitive code can be minimised. Once defined, a function can be used again and again anywhere.
- **Makes debugging easy:** Functions divide the code into various fragments. It helps in debugging as now the error is confined to a small fragment and not in the entire code.

Scope of variables

A scope is a region of the program and broadly speaking there are three places, where variables can be declared:

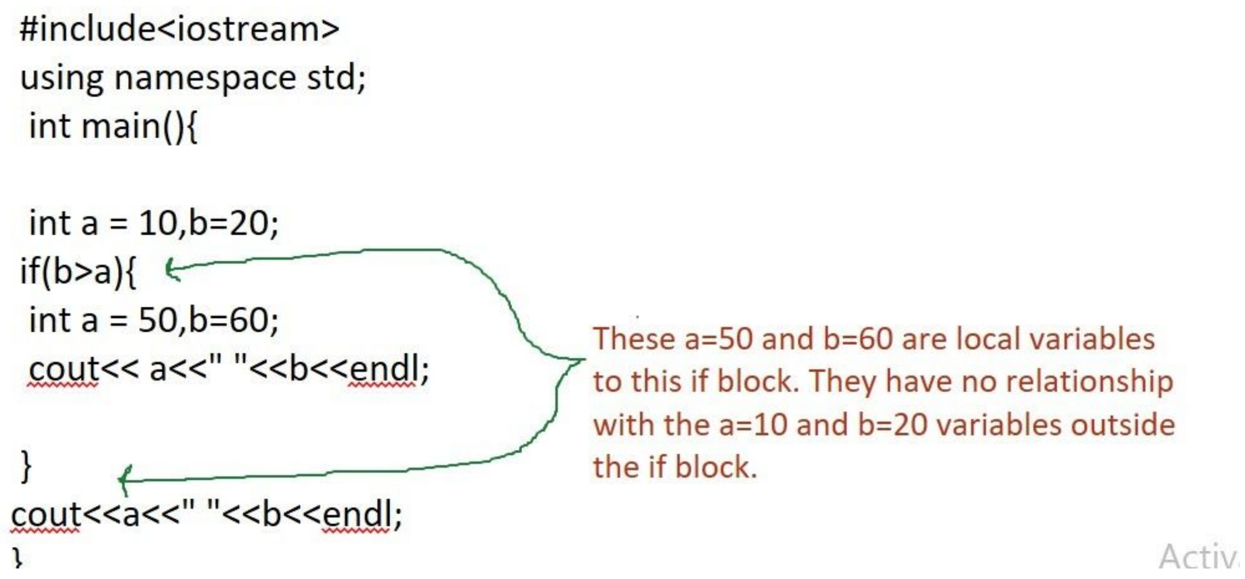
1 A variable can be declared inside an if block, for block or inside a function. In this case we say that it is a local variable.

2 We can also define a variable in the definition of function parameters.

3 The variables which are declared outside of all the functions are called global variables.

Local variables:

Variables that are declared inside a function or block are local variables. They can be used only by statements that are inside that function or block of code.
Let's see this example:



```
#include<iostream>
using namespace std;
 int main(){

 int a = 10,b=20;
if(b>a){
 int a = 50,b=60;
 cout<< a<<" "<<b<<endl;

}
cout<<a<<" "<<b<<endl;
}
```

These a=50 and b=60 are local variables to this if block. They have no relationship with the a=10 and b=20 variables outside the if block.

The output will be:

```
 50 60
 10 20
```

Note: The argument variables of a function are local to that function only. You can't access them outside the function. Also notice one thing in the above example that you are declaring variables of the same name, why isn't this code giving any errors. This code will not give any error because

we are declaring the variables of same name into different scopes. We can't declare variables of the same name in the same scope.

Global variables:

Variables which are defined outside all the functions, usually on the top of code are called global variables. They can be accessed by any function.

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

// Global variable declaration:
int g;

int main () {
  // Local variable declaration:
  int a, b;

  // actual initialization
  a = 10;
  b = 20;
  g = a + b;

  cout << g;

  return 0;
}
```

In the above code snippet g is the global variable.

When a local variable is defined, it is not initialized by the system, you must initialize it yourself. Global variables are initialized automatically by the system.

Data Type	Initializer
int	0
char	'\0'

float	0
double	0
pointer	NULL

Question 4
Will the following code generate any error?

```
#include <iostream>
using namespace std;
void functional(int a) {
    int b = a;
    b = b *10;
}

int main() {
    int a = 10;
    functional(a);
    cout << b << endl;

}
```

Yes the above code will give an error as we are trying to access variable b outside the function. Variable b is local to the function.

<u>Passing variables to functions:</u>

There are two ways of passing variables to a function. One is pass by value and the other is pass by reference. Here we will study pass by value.

Pass by value:

When we pass variables by value, then the formal parameters are allocated to a new memory. They have the same values as the actual parameters. We can also say that we are passing a copy of the variables to the function. Any changes in the formal parameters don’t affect the actual parameters because formal parameters are allocated a new memory.

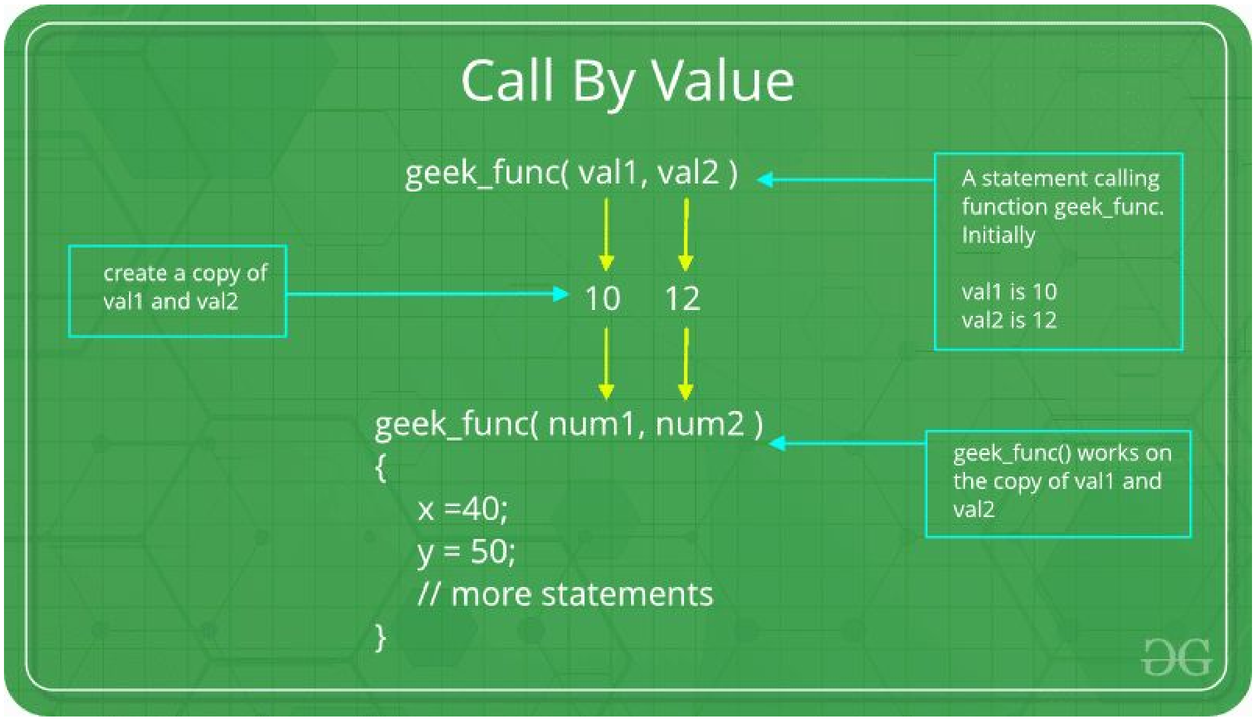
Example :

```
#include<iostream>
using namespace std;
void  change (int x, int y){

 //x and y are formal    parameters
x=x+5;
y =y*3;
cout<<x<<": "<<y<<endl;
}

int main () {
Int a=15, b=27;
change (a, b);
//    a and b are actual parameters
cout<<a<<": "<<b;
}

```

Output:

```
20: 81

15: 27
```

The below figure will help you in understanding :


Call By Value
geek_func(val1, val2)
A statement calling function geek_func. Initially
val1 is 10
val2 is 12
create a copy of val1 and val2
10
12
geek_func(num1, num2)
{
x =40;
y = 50;
// more statements
}
geek_func() works on the copy of val1 and val2

Question 5

What will be the output of this code snippet?

```
int func(int a){

   a += 10;

   }

int main() {

   int a = 5;

   func(a);

   cout << a;

}
```

The output of this code snippet is 5. As we have studied the change in formal parameters doesn't affect the actual parameters, so variable a defined in main function is not affected by the change in variable a of function func.

Introduction to pointers :

As you know every variable is a memory location and every memory location has its address defined which can be accessed using ampersand (&) operator which denotes an address in memory. Consider the following which will print the address of the variables defined −

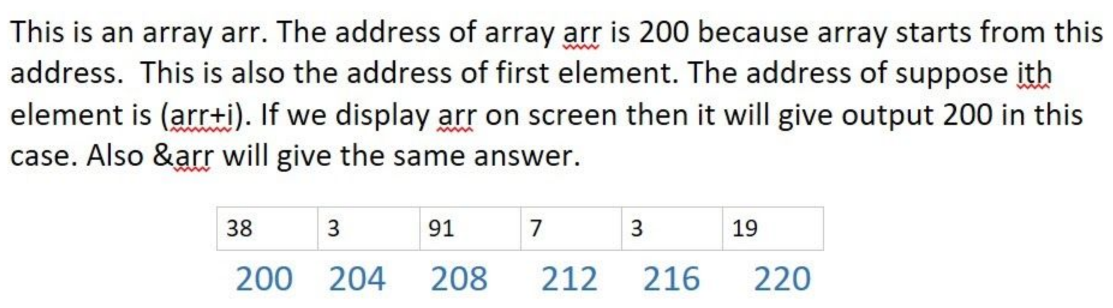
ideone link

```
#include <iostream>
using namespace std;
int main () {
  int  var1;
  char var2[10];
  cout << "Address of var1 variable: ";
  cout << &var1 << endl;

  cout << "Address of var2 variable: ";
  cout << &var2 << endl;

  return 0;
}
```

When the above code is compiled and executed, it produces the following result −

Address of var1 variable: 0xbfebd5c0

Address of var2 variable: 0xbfebd5b6



This is an array arr. The address of array arr is 200 because array starts from this address. This is also the address of first element. The address of suppose ith element is (arr+i). If we display arr on screen then it will give output 200 in this case. Also &arr will give the same answer.

38	3	91	7	3	19
200	204	208	212	216	220

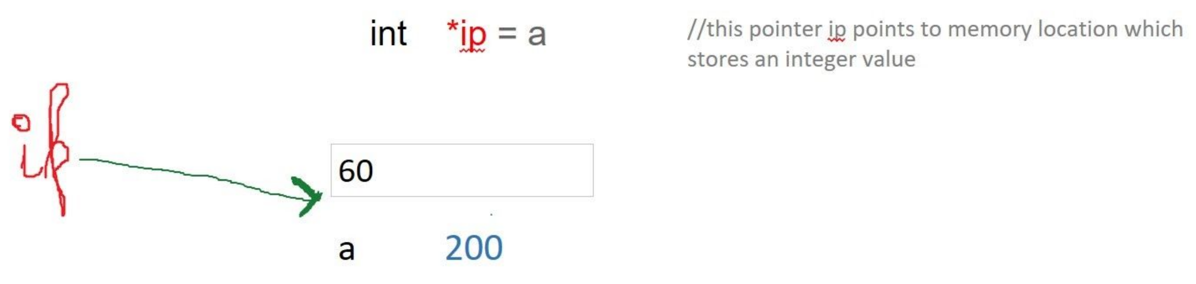
What are Pointers?

A pointer is a variable whose value is the address of another variable. Like any variable or constant, you must declare a pointer before you can work with it. The general form of a pointer variable declaration is −

type *var-name;

Here, type is the pointer's base type; it must be a valid C++ type and var-name is the name of the pointer variable. The asterisk(*) you used to declare a pointer is the same asterisk that you use for multiplication. However, in this statement the asterisk is being used to designate a variable as a pointer. Following are the valid pointer declaration −

```
int    *ip;           // pointer to an integer

double *dp;       // pointer to a double

float  *fp;        // pointer to a float

char   *ch        // pointer to character
```


int *ip = a
//this pointer ip points to memory location which stores an integer value
ip
60
a
200

Suppose variable a of integer datatype is at memory location 200 and a stores the value 60

The actual data type of the value of all pointers, whether integer, float, character, or otherwise, is the same, a long hexadecimal number that represents a memory address. The only difference between pointers of different data types is the data type of the variable or constant that the pointer points to.

Using Pointers in C++

There are few important operations, which we will do with the pointers very frequently.

(a) We define a pointer variable.

(b) Assign the address of a variable to a pointer.

(c) Finally access the value at the address available in the pointer variable. This is done by using unary operator * that returns the value of the variable located at the address specified by its operand.

Following example makes use of these operations −

Example :-

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

```
int main () {
  int  var = 20;    // actual variable declaration.
  int  *ip;         // pointer variable

  ip = &var;        // store address of var in pointer variable
  cout << "Value of var variable: ";
   cout << var << endl;

  // print the address stored in ip pointer variable
  cout << "Address stored in ip variable: ";
  cout << ip << endl;

  // access the value at the address available in pointer
  cout << "Value of *ip variable: ";
  cout << *ip << endl;
  return 0;
}
```

When the above code is compiled and executed, it produces result something as follows −

```
Value of var variable: 20

Address stored in ip variable: 0xbfc601ac

Value of *ip variable: 20
```

<u>Get the value from the address using pointers</u>:

ProgramLink: sapphire engine link, ideone link

<u>Program:</u>

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

```

```cpp
int main()
{
      int *ptr, var;
    cout << "Enter an integer: " << endl;
    cin >> var;

      ptr = &var; // assign address of var to ptr
      cout << "Address of var = " << &var << endl;
      cout << "Address assigned to ptr = " << ptr << endl;
      cout << "var = " << var << "\n*ptr = " << *ptr << endl; //
access value pointed by ptr

      return 0;
  }
```

Standard Output

```
Enter an integer:
12
Address of var = 0x7ffd9004121c
Address assigned to ptr = 0x7ffd9004121c
var = 12
*ptr = 12
```

Explanation: In the above program, we declared two variables, viz., **ptr** and **var**. **ptr** is a pointer to an integer whereas var is variable of type integer. Then the program prompts the user to enter the value of **var**. Once the user enters the value of **var**, the address of variable **var** is assigned to **ptr**. To access the value pointed to by a pointer variable, we use “*” operator(also known as dereference operator or value at address operator).Thus, when we write ***ptr** in the above program, we are getting the value stored at the location that is pointed by the pointer variable **ptr**.In the above example, this value is 12.

Call-by-reference:

ProgramLink: sapphire engine link, ideone link

Program:

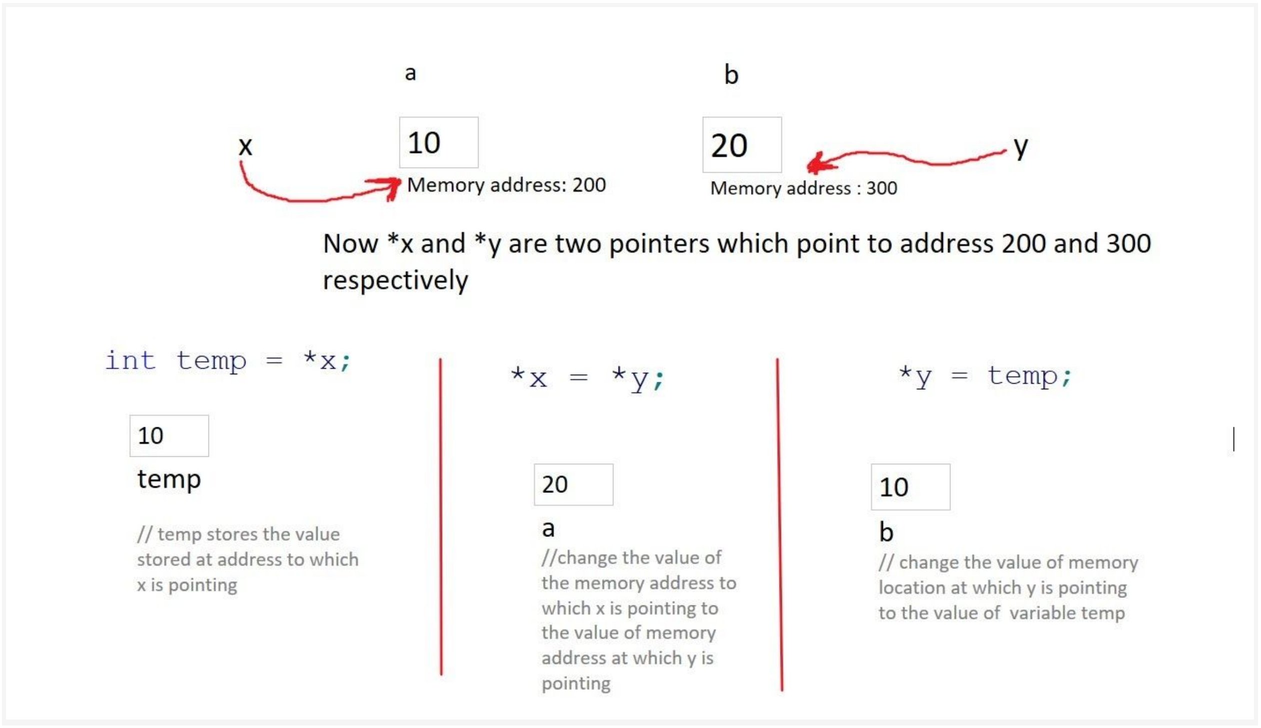
```
#include <iostream>
using namespace std;

void swap(int *, int *); // function prototype
void swap(int *x, int *y) { // function definition
   int temp = *x;
   *x = *y;
   *y = temp;
}
int main()
{
   int a, b;
    cout << "Enter a and b: " << endl;
    cin >> a >> b;
    cout << "Before swapping a = " << a << " and b = " << b <<
endl;
    swap(&a, &b); // call by reference
    cout << "After swapping a = " << a << " and b = " << b <<
endl;
   return 0;
}
```

Standard Output

```
Enter a and b:
10 20
Before swapping a = 10 and b = 20
```

```
After swapping a = 20 and b = 10
```

Explanation: In this program, after reading the values from the user. We swap the two values using the function **swap()**. This function requires two arguments that are two integer pointers. When we invoke the function swap(), we pass the address of variables a and b. This is known as call-by-reference. In the definition of swap(), we swap the contents of named-locations **a** and **b** by using another variable(in this program temp).


a
b
x
10
Memory address: 200
20
Memory address : 300
y
Now *x and *y are two pointers which point to address 200 and 300 respectively
int temp = *x;
10
temp
// temp stores the value stored at address to which x is pointing
*x = *y;
20
a
//change the value of the memory address to which x is pointing to the value of memory address at which y is pointing
*y = temp;
10
b
// change the value of memory location at which y is pointing to the value of variable temp

Here are some questions on pointers which will help you in understanding pointers and arrays.

Question 1 Find the output

```
#include <bits/stdc++.h>
#include <iostream>
using namespace std;

```

```
int main()
{
int a = 7;
int *c = &a;
 c=c+1;
cout  << a << "  " << c << endl;
cout<<*c<<endl;
            return 0;
}
```

output

```
7  0x7ffdaf1a5868

1415355648
```

In the above code snippet, we are declaring an integer variable a and initialising it with value 7. Then we are declaring a pointer variable which stores the address of a. Notice that here we are doing arithmetic addition of pointers. Like we increment the variables, we can increment pointer variables also. If we increment a pointer, it means the pointer variable now points to the next memory location. Suppose pointer ptr initially points to a memory location 200 and the next memory address is 204, then if we do ptr=ptr+1, ptr will now point to memory location 204. The new memory location may contain some garbage value.

Here ‘a’ stores the value 7. Pointer c stores the address of a . Now incrementing the value of c, it now points to a different memory address. Displaying c on screen will give another memory address. This memory address may have some garbage value which is printed on screen using *c. The garbage value is `1415355648`.

Question 2   Find the output

```
#include <bits/stdc++.h>
#include <iostream>
using namespace std;
```

```cpp

int main()
{
int a = 7;
int *c = &a;
c = c + 3;
cout  << c << endl;
      return 0;
}
```

Output

```
0x7ffc0a59b820
```

The explanation is same as above

Question 3   Find the output

```cpp
#include <bits/stdc++.h>
#include <iostream>
using namespace std;

int main()
{
int a[] = {1, 2, 3, 4};
cout << *(a) << " " << *(a+1);
cout << a <<"   "<< &a;
      return 0;
}
```

Standard Output

```
1 2
0x7fff405184f0   0x7fff405184f0
```

The name of the array signifies the address of the first element of the array, it also signifies the address of that array. That’s why cout<<a and cout<<&a fives the same memory address. *a gives the element at index 0, so cout<<*a gives 1 and *(a+1) gives the element at memory address(a+1) i.e at index 1 which is 2.

Question 3 Find the output

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

int main()
{
int a[6] = {1, 2, 3};
int *b = a;
cout << b[2];
      return 0;
}
```

Standard Output

```
3
```

In the above code snippet we make a pointer variable b which stores the address of array a. Now b[2] effectively means *(b+2) i.e *(a+2) because b stores the address of a.

Question 4 Find the output

```
#include <bits/stdc++.h>
#include <iostream>
using namespace std;

int main()
{
int a[] = {1, 2, 3, 4};
int *p = a++;
cout << *p << endl;
      return 0;
}
```

Output: error

We cannot change the address of an array. Writing a++ means a=a+1, This means we are shifting the entire array's address by 4 bytes. This is not a valid operation.

Question 5 Find the output

```
#include <bits/stdc++.h>
#include <iostream>
using namespace std;

int main()
{
char ch = 'a';
char* ptr = &ch;
ch++;
cout << *ptr << endl;
      return 0;
}
```

Standard Output

```
b
```

As we have studied earlier, we can do arithmetic operations on characters. This will do the same work as we do arithmetic operations on their ASCII values.So ch++ means now the character is b. We have also declared the pointer variable ptr which stores the address of variable ch. By writing ch++ we have changed the content of variable ch to ‘b’. Now if we access that memory location by pointer ptr, it will give us the value ‘b’.