Click to prove you're human

Functions may be return type functions and non-return type functions. The non-return type functions do not return any value to the calling function; the type of such functions is void. These functions may or may not have any argument to act upon. A few illustrations of such functions are given below.void Write (void) { printf("You need a compiler for learning C language.");}The first line in the above definition may also be written asvoid Write ()Program presents an example where a void function is defined to display a message.Illustrates a void function with void parameter list.#include void Display(void); //Function prototype, void parameter list void main() { Display(); //the function call } //end of main function void Display() { // Definition of Display printf("Play an outdoor game for better health."); printf("Also remember \"It is practice which makes a man/woman perfect in programming in C.\""); }The expected output of the above program is as given below. A function is a block of code which only runs when it is called.You can pass data, known as parameters, into a function.Functions are used to perform certain actions, and they are important for reusing code: Define the code once, and use it many times. Predefined Functions So it turns out you already know what a function is. You have been using it the whole time while studying this tutorial! For example, main() is a function, which is used to execute code, and printf() is a function; used to output/print text to the screen: int main() { printf("Hello World!"); return 0;}Try it Yourself Create a FunctionTo create (often referred to as declare) your own function, specify the name of the function, followed by parentheses () and curly brackets {}: void myFunction() { // code to be executed}Example ExplainedmyFunction() is the name of the functionvoid means that the function does not have a return value. You will learn more about return values later in the next chapterInside the function (the body), add code that defines what the function should do Call a FunctionDeclared functions are not executed immediately. They are "saved for later use", and will be executed when they are called.To call a function, write the function's name followed by two parentheses () and a semicolon ;In the following example, myFunction() is used to print a text (the action), when it is called: Inside main, call myFunction(): // Create a functionvoid myFunction() { printf("I just got executed!");}int main() { myFunction(); // call the function return 0;}// Outputs "I just got executed!"Try it Yourself A function can be called multiple times: void myFunction() { printf("I just got executed!");}int main() { myFunction(); myFunction(); myFunction(); return 0;}// I just got executed!// I just got executed!// I just got executed!Try it Yourself Calculate the Sum of NumbersYou can put almost whatever you want inside a function. The purpose of the function is to save the code, and execute it when you need it.Like in the example below, we have created a function to calculate the sum of two numbers. Whenever you are ready to execute the function (and perform the calculation), you just call it: void calculateSum() { int x = 5; int y = 10; int sum = x + y; printf("The sum of x + y is: %d", sum);}int main() { calculateSum(); // call the function return 0;}// Outputs The sum of x + y is: 15Try it Yourself This was just an example to demonstrate a simple function with different statements in C. The real power of a function is revealed in the next chapter, when we pass "parameters" to it.This allows the function to calculate the sum of any numbers, instead of being limited to the fixed values 5 and 10. In C programming, a function that does not take any arguments and does not return a value is called a void function. The syntax for defining a void function is as follows: Syntax: return_type function_name (parameter1, parameter2, ...) { // body of Statement ; } Example : void function_name ( ) { // body of Statement ; } Here, the 'void' keyword is used as the return type to indicate that the function does not return any value.A void function can be called in the same way as other functions, by simply using the function name followed by empty parentheses: function_name ( );Here is an example of a void function that prints a message to the console: This program defines a void function called add that simply prints the message "Total of a and b" to the console. The main function calls the add function, which executes the code inside the function and prints the message.Void functions are useful in C programming when you want to perform a specific task without returning any value, such as displaying a message, initializing a variable or updating a data structureSource Code //No Return Without Argument Function in C/* 1.Function Declaration 2.Function Definition 3.Function Calling*/#include//Function Declarationvoid add();int main(){ //Function Calling add(); return 0;}//Function Definitionvoid add(){ int a,b,c; printf("Enter The Value of A & B :"); scanf("%d%d",&a,&b); c=a+b; printf("Total : %d",c);} To download raw file Click HereOutputEnter The Value of A & B :1234Total : 46 Prerequisite: Functions in CA function in C can be called either with arguments or without arguments. These functions may or may not return values to the calling functions. All C functions can be called either with arguments or without arguments in a C program. Also, they may or may not return any values. Hence the function prototype of a function in C is as below:Call by ValueCall by value in C is where in the arguments we pass value and that value can be used in function for performing the operation. Values passed in the function are stored in temporary memory so the changes performed in the function don't affect the actual value of the variable passed. Example: C // C Program to implement// Call by value#include // Call by valueint sum(int x, int y){ int c; c = x + y; // Integer value retured return c;} // Driver Codeint main(){ // Integer Declared int a = 3, b = 2; // Function Called int c = sum(a, b); printf("Sum of %d and %d : %d", a, b, c); return 0;}Call by ReferenceCall by reference is the method in C where we call the function with the passing address as arguments. We pass the address of the memory blocks which can be further stored in a pointer variable that can be used in the function. Now, changes performed in the values inside the function can be directly reflected in the main memory.Example: C // C Program to implement// Call by referencevoid swap(int* x, int* y){ int temp = *x; *x = *y; *y = temp;} // Driver Codeint main(){ // Declaring Integer int x = 1, y = 5; printf("Before Swapping: x:%d , y:%d", x, y); // Calling the function swap(&x, &y); printf("After Swapping: x:%d , y:%d", x, y); return 0;}OutputBefore Swapping: x:1 , y:5After Swapping: x:5 , y:1Types of Function According to Arguments and Return ValueFunctions can be differentiated into 4 types according to the arguments passed and value returns these are:Function with arguments and return valueFunction with arguments and no return valueFunction with no arguments and with return valueFunction with no arguments and no return value1. Function with arguments and return valueSyntax:Function declaration : int function (int);Function call : function(x);Function definition: int function(int x) { statements; return x; } Example: C // C code for function with arguments// and return value#include #include int function(int, int[]); int main(){ int i, a = 20; int arr[5] = { 10, 20, 30, 40, 50 }; a = function(a, &arr[0]); printf("value of a is %d", a); for (i = 0; i < 5; i++) { printf("value of arr[%d] is %d", i, arr[i]); } return 0;} int function(int a, int* arr){ int i; a = a + 20; arr[0] = arr[0] + 50; arr[1] = arr[1] + 50; arr[2] = arr[2] + 50; arr[3] = arr[3] + 50; arr[4] = arr[4] + 50; return a;}Outputvalue of a is 40value of arr[0] is 60value of arr[1] is 70value of arr[2] is 80value of arr[3] is 90value of arr[4] is 1002. Function with arguments but no return valueWhen a function has arguments, it receives any data from the calling function but it returns no values. These are void functions with no return values.Syntax:Function declaration : void function (int);Function call : function(x);Function definition: void function(int x) { statements; } Example: C // C code for function// with argument but no return value#include void function(int, int[], char[]); int main(){ int a = 20; int ar[5] = { 10, 20, 30, 40, 50 }; char str[30] = "geeksforgeeks"; // function call function(a, &ar[0], &str[0]); return 0;} void function(int a, int* ar, char* str){ int i; printf("value of a is %d", a); for (i = 0; i < 5; i++) { printf("value of ar[%d] is %d", i, ar[i]); } printf("value of str is %s", str);}Outputvalue of a is 20value of ar[0] is 10value of ar[1] is 20value of ar[2] is 30value of ar[3] is 40value of ar[4] is 50value of str is geeksforgeeks3. Function with no argument and no return valueWhen a function has no arguments, it does not receive any data from the calling function. Similarly, when it does not return a value, the calling function does not receive any data from the called function.Syntax:Function declaration : void function();Function call : function();Function definition : void function() { statements; } Example: C // C code for function with no// arguments and no return value#include void value(void); void main() { value(); } void value(void){ float year = 1, period = 5, amount = 5000, inrate = 0.12; float sum; sum = amount; while (year #include <conio.h> int sum(int n); void main() { int n, x; printf("Enter the number:n"); scanf("%d", &n); x = sum(n); printf("nThe sum of %d number is =%d", n, x); getch(); } int sum(int n) { int i, num, s = 0; for (i = 1; i <= n; i++) { printf("Enter the number: "); scanf("%d", &num); s = s + num; } return s; }The voidstatement is used in the function when there is no need to return the value. It means this function does not give output to the called function. Though the function is return type void it may contain a return keyword to transfer the control at end of the function.void function_name(list_of_parameters);The user-defined function can be classified into four categories on the basis of the arguments and return value.Function with no arguments and no return value.Function with no argument and a return value.Function with argument and no return value.Function with argument and a return value.This type of function will not return any value to the calling function so we use return type void and it will not also pass value to the called function on the argument whether kept empty or use void. The syntax is as: void func (void); //return type is void and argument also void main() { func(); //function call .. } void func(void) //function defination { statement; } // program of user-defined function with no argument and no return value in C programming #include <stdio.h> #include <conio.h> void sum(void); void main() { sum(); getch(); } void sum(void) { int sum, a, b; printf("Enter first number:"); scanf("%d", &a); printf("Enter second number:"); scanf("%d", &b); sum = a + b; printf("The sum of two number is %d", sum); }This type of function does not receive any arguments but can return a value. Here it will not pass the argument but it can return value. So we used int as the return type it will return an integer value. The syntax is as: int func(void); //function declaration with no argument but return type main() { int r; r=func(); //calling function . } int func(void) { .. .. return (expression); } // program of user-defined function with no argument and return value in C programming #include <stdio.h> #include <conio.h> int sum(void); void main() { printf("The sum of two number is %d", sum()); getch(); } int sum(void) { int sum, a, b; printf("Enter first number:"); scanf("%d", &a); printf("Enter second number:"); scanf("%d", &b); sum = a + b; return sum; }This type of function can pass the argument but will not return value to the main() function. Here we use void as the return type and value can be pass from the main() function to the called function. The syntax is as: void func(int,int); //function declaration with no return value but argument main() { func (a,b); //calling function . } void func (int c, int d) //function definition { . statemnts . } // program of user defined function with argument and no return value in C programming #include <stdio.h> #include <conio.h> void sum(int a, int b); void main() { int a, b; printf("Enter first number: "); scanf("%d", &a); printf("Enter second number: "); scanf("%d", &b); getch(); } void sum(int a, int b) { int s; s = a + b; printf("The sum of two number is %d", s); }This type of function will pass the value from the main program to the function and the calculated value will be a return to the main function. So we use the int as the return type and pass the argument. The syntax is as: int func(int, int); //function declaration with return type and argument main() { int r; r=func(a,b); //calling function .. func (c,d); } int func (int a, int b) //function definition { .. . return (expression); } // program of user defined function with argument and return value in C programming #include <stdio.h> #include <conio.h> int sum(int a, int b); void main() { int a, b, s; printf("Enter first number: "); scanf("%d", &a); printf("Enter second number: "); scanf("%d", &b); s = sum(a, b); printf("The sum of two number is %d", s); getch(); } int sum(int a, int b) { int x; x = a + b; return x; }The variables that are defined within the body of a function or a block are local to that function or block only and are called local variables. For example: void func(); { int a,b,c; .. }The local variables can be used only in those functions or blocks, in which they are declared. The same name variables may be used in different functions.The variables that are defined outside any function are called global variables. All functions in the program can access and modify global variables. It is useful to declare a variable global if it is to be used by many functions in the program. Global variables are automatically initialized to 0 at the time of declaration. int a=10, b=20; //declaration of global variable before main() function void main() { Local variable; .. }Static variables are declared by writing keyword static in from of the declaration.static data_type var_name;A static variable is initialized only once and the value of a static variable is retained between function calls. If static variables are not initialized then it is automatically initialized to 0.Recursion is a powerful technique for writing complicated problems in an easy way. According to this technique, a problem is defined in terms of itself. The problem is solved by diving it into smaller problems, which are similar in nature to the original problem. These smaller problems are solved and their solutions are applied to get the final solution of our original problem. Therefore, the function which calls itself is known as a recursive function. Void (NonValue-Returning) functions:Void functions are created and used just like value-returning functions except they do not return a value after the function executes. In lieu of a data type, void functions use the keyword "void." A void function performs a task, and then control returns back to the caller--but, it does not return a value. You may or may not use the return statement, as there is no return value. Even without the return statement, control will return to the caller automatically at the end of the function. A good utilization of a void function would be to print a header/footer to a screen or file. Remember: there are two kinds of subprograms that the C++ language utilizes: value-returning functions and void functions. Both value-returning functions and void functions receive values through their parameter lists. A value-returning function can only return one value to the calling environment. The caller invokes (calls) a value-returning function by using its name and argument list in an expression (i.e., 1. assignment, 2. output, or as an 3. argument in another function call): //value-returning function call (assignment):y = 2.0 * sqrt(x); In contrast, a void function (method or procedure, in other languages) does not return a function value. Nor is it called from within an expression. Instead, the function call appears as a complete, stand-alone statement. One example is the get function associated with the istream and ifstream classes: //void function call:cin.get(); Another example://void function call:printName(name); Bjarne Stroustrup's C++ Glossary Similarities Between Value-Returning and Void (NonValue-Returning) functions:Both: require function definitions (i.e., headers and bodies)Both: definitions can be placed before or after function main()... though, if placed after main() function, prototypes must be placed before main()Both: formal parameter list can be empty--though, parentheses still requiredBoth: actual parameter list can use expression or variable, but must match in "TON": type, order, number Differences Between Value-Returning and Void (NonValue-Returning) functions:Void function: does not have return typeUses keyword void in function headerCall to void function is stand-alone statement //Void (NonValue-returning) function definition syntax: including header and bodyvoid functionName(formal parameter list) //function header{ //function body statements... //void (nonvalue-returning) function can still use return statement //but, does not return any values... return;} //function call syntax:functionName(actual parameter list); //stand-alone statement only Call to void function with empty formal parameter list: functionName(); //stand-alone statement only ***Void function calls can ONLY be used w/in stand-alonestatements. ***Conversely, value-returning function calls can be used in output: e.g., cout statementsassignmentarguments in other function calls Function Parameters Function Parameter Types: Two Types of Function Parameters:Value Parameter: formal parameter receives copy of content of corresponding actual parameterReference Parameter: formal parameter receives location (memory address) of corresponding actual parameter Using Value and Reference Parameters:Value Parameters:Value of corresponding actual parameter copied into formal parameterValue parameter has own copy of dataDuring program execution, value parameter manipulates data stored in own memory spaceValue parameters will accept expressions, constants, or variables from actual parameters (i.e., in function call) Void Function Definition Using Value Parameters Example://Void (NonValue-returning) function definition syntax: including header and bodyvoid functionName(dataType variable, dataType variable) //function header{ //function body statements... //void (nonvalue-returning) function can still use return statement //but, does not return any values... return;} Reference Parameters:Formal parameter receives and stores address of corresponding actual parameterDuring program execution, address stored in reference parameter can modify data of corresponding actual parameter, by sharing same memory spaceReference parameters can: pass one or more values from function, and can change value of actual parameterReference parameters useful in three situations: Returning more than one value Changing value of actual parameter When passing address would save memory space and time ***Reference parameter limitations: reference parameters will not accept expressions or constants from actual parameters (i.e., in function call), ONLY variables Void Function Definition Using Reference Parameters Example://Void (NonValue-returning) function definition syntax: including header and bodyvoid functionName(dataType& variable, dataType& variable) //function header{ //function body statements... //void (nonvalue-returning) function can still use return statement //but, does not return any values... return;} Function Calls Using Value and Reference Parameter Examples:Void function call using value parameters (can use expression, constant, or variable)://Void (NonValue-returning) function call with arguments functionName(expression or constant or variable, expression or constant or variable); //stand-alone statement only Void function call using reference parameters (can NOT use expression or constant, ONLY variables)://Void (NonValue-returning) function call with arguments functionName(variable, variable); //stand-alone statement only Function Parameters And Memory Allocation When a function is called:Memory for formal parameters (in header) and (local) variables declared in body of function allocated in function data areaValue parameter: value of actual parameter copied into memory cell of its corresponding formal parameterReference parameter: address of actual parameter passed to formal parameter (content of formal parameter is an address) During execution, changes made by formal parameter permanently change value of actual parameter Stream variables (e.g., ifstream and ofstream) should be passed by reference to function Summary of pass by value vs. pass by reference:Pass by value: Makes copy of actual parameterPasses copy of contentsOriginal variable's contents DO NOT changeHow? Do nothing. Default.Safer Pass by reference:Passes address of actual parameterAccesses original variable's contents (via address)Original variable's contents DO changeHow? Add ampersand (&) before parameter name in header and prototypeMore efficientNote: Although the C++ language allows value-returningfunctions to have both value parameters and reference parameters, itis not recommended. By definition, a value-returningfunction returns a single value; this value is returned via the returnstatement. Use void functions with reference parameters to return(modify) more than one value. Program demos void functions using value and reference parameters:/*This program demos void functions using value parameters vs. reference parametersReference Parameter:If a formal parameter is a reference parameter, itreceives the address of the corresponding actualparameterA reference parameter stores the address of thecorresponding actual parameterDuring program execution to manipulate the data,the address stored in the reference parameter directsit to the memory space of the corresponding actualparameter*///header files#includeusing std::cout;using std::cin;using std::endl;//prototype functionsvoid readDate(int& month, int& day, int& year); //reference parametersvoid printDate(int month, int day, int year); //value parameters//initialize constants//initialize global variablesint main(){ //initialize automatic variables int mm, dd, yy; readDate(mm, dd, yy); printDate(mm, dd, yy); cout month >> ch >> day >> ch >> year;}void printDate(int month, int day, int year){ cout