 Functions

Look at this Powerpoint @ <http://www.slideshare.net/kumar_vic/function-in-c-30522092>

You can define functions and input defined variables in the parenthesis of the function. You can then call function from the name you gave them and input the values of those variables in its parenthesis or parameters. Full fledged functions like int main has datatypes and names made by the user to identify them. Like variables functions can be called w/o being defined again. All they need is their name and their (). The return type gives the final calculation of the function.

<https://www.thenewboston.com/videos.php?cat=16&video=17486>

<https://www.thenewboston.com/videos.php?cat=16&video=17487>

**A program may be broken up into manageable functions**

*Code reuse* refers to writing the code to perform a task once and then reusing it each time you need to perform the task

**A function call is a statement that causes a function to execute. A function definition contains the statements that make up the function**

When making a function, you have to *define* it first. You must write its definition, which consists of writing the return type, name, parameter list, and body of the function.

*Return type*: A function can send a value to the part of the program that executed it. The return type is the data type of the value sent to it. It gives the final calculation of the function. Not all functions need a return type. Void functions don't need a return type.

*Parameter list*: The program can send data into a function. The parameter list is a list of variables that hold the values being passed to the function.

A function is executed when it's *called*. All functions except function main must be called before it's executed. When a f(x) is called, the program branches to that function and executes the statements in its body.

<https://www.thenewboston.com/videos.php?cat=16&video=17487>

// This program has two functions: main and displayMessage  
#include <iostream>  
using namespace std;

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
// Definition of function displayMessage \*  
// This function displays a greeting. \*  
//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void displayMessage()  
{  
 cout << "Hello from the function displayMessage.\n";  
}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
// Function main \*  
//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

int main()  
{  
 cout << "Hello from main.\n";  
 displayMessage();  
 cout << "Back in function main again.\n";  
 return 0;  
}

Like variables functions can be called w/o being defined again. All they need is their name and their (). The return type gives the final calculation of the function.

**A function prototype eliminates the need *to place a function definition before all calls to the function***

EX: void displayMessage(); // at least datatypes should be inside prototype

You must either place the f(x) definition 1st or place the function prototype first. You can't type both for 1 particular function. If you do the program will not compile.

*What about the return type? Where is it in a function prototype? In a void function the keyword void is the return type of the function. The return type is in the function definition that occurs later in the program the prototype is in.*

**When a function is called, the program may send values into the function.**

Values that are sent into a f(x) are called *arguments*.

EX: pow(2.0, 4.0);

Variables that are sent into a f(x) are called *parameters*. The scope of a parameter is limited to the body of the f(x that uses it. The parameter has a local scope

EX: void displayValue(int num)

{

cout << "The value is " << num << endl;

}

Any argument listed inside the parenthesis of a function call is copied into the function's parameter variable

<https://www.thenewboston.com/videos.php?cat=16&video=17487>

*Can variables be defined when functions are called? Can they only be defined when the function is being defined?*

**When an argument is passed into a parameter, only a copy of the argument's value is passed. Changes to the parameter don't affect the original argument.**

**Any changes you make to the parameter in the function is made to the copy and does not change the value of the argument in the calling function.**

From <[*https://elearning.utdallas.edu/bbcswebdav/pid-894609-dt-content-rid-7979608\_1/users/dgv130030/CS%201136%20F15/Lesson%209/CS%201136%20Laboratory%20Lesson%209.pdf*](https://elearning.utdallas.edu/bbcswebdav/pid-894609-dt-content-rid-7979608_1/users/dgv130030/CS%201136%20F15/Lesson%209/CS%201136%20Laboratory%20Lesson%209.pdf)>

<https://elearning.utdallas.edu/bbcswebdav/pid-894609-dt-content-rid-7979608_1/users/dgv130030/CS%201136%20F15/Lesson%209/CS%201136%20Laboratory%20Lesson%209.pdf>

When only a copy of an argument is passed to a f(x), it's said to be *passed by value.*

**Functions are ideal for use in menu-driven programs. When the user selects an item from a menu, the program can call the appropriate function.**

*Menu-driven program*: program execution is controlled by a user selecting from a list of actions.

*Menu*: list of choices on the screen

A *modular* program is broken up into functions that perform specific tasks.

**The return causes a function to end immediately**

// This program uses a function to perform division. If division  
// by zero is detected, the function returns.  
#include <iostream>  
using namespace std;

// Function prototype.  
void divide(double, double);

int main()  
{  
 double num1, num2;

cout << "Enter two numbers and I will divide the first\n";  
 cout << "number by the second number: ";  
 cin >> num1 >> num2;  
 divide(num1, num2);  
 return 0;  
}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
// Definition of function divide. \*  
// Uses two parameters: arg1 and arg2. The function divides arg1\*  
// by arg2 and shows the result. If arg2 is zero, however, the \*  
// function returns. \*  
//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void divide(double arg1, double arg2)  
{  
 if (arg2 == 0.0)  
 {  
 cout << "Sorry, I cannot divide by zero.\n";  
 return;  
 }  
 cout << "The quotient is " << (arg1 / arg2) << endl;  
}

**A function may send a value back to the part of the program that called the function**

Functions that return a value are appropriately known as *value-returning functions.*

return expression;

Expression is the value, variable or named constant that holds a value to be returned. The value of the expression is converted to the data type that the function returns, and is sent back to the statement that called the f(x).

A value-returning f(x) returns a value of a specific data type.

*How is like and different from the auto datatype?*

"The **auto** keyword directs the compiler to use the initialization expression of a declared variable, or lambda expression parameter, to deduce its type"

From <[*https://msdn.microsoft.com/en-us/library/dd293667.aspx*](https://msdn.microsoft.com/en-us/library/dd293667.aspx)>

<https://www.thenewboston.com/videos.php?cat=16&video=17504>

**Functions may return true or false values**

// This program uses a function that returns true or false.  
#include <iostream>  
using namespace std;

// Function prototype  
bool isEven(int);

int main()  
{  
 int val;

// Get a number from the user.  
 cout << "Enter an integer and I will tell you ";  
 cout << "if it is even or odd: ";  
 cin >> val;  
   
 // Indicate whether it is even or odd.  
 if (isEven(val))  
 cout << val << " is even.\n";  
 else  
 cout << val << " is odd.\n";  
 return 0;  
}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
// Definition of function isEven. This function accepts an \*  
// integer argument and tests it to be even or odd. The function \*  
// returns true if the argument is even or false if the argument \*  
// is odd. The return value is an bool. \*  
//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

bool isEven(int number)  
{  
 bool status;

if (number % 2 == 0)  
 status = true; // number is even if there is no remainder.  
 else  
 status = false; // Otherwise, the number is odd.  
 return status;  
}

**A local variable is defined inside a function & isn't accessible outside the function. A global variable is defined outside all functions & is accessible to all functions in its scope.**

A function's local variables exist only while the functions is executing. This is known as the *lifetime* of a local variable. This means that any value stored in a local variable is lost between calls to the f(x) in which the variable is declared.

*What is the lifetime of a global variable?*

They are any variables defined outside all the f(X)s in a program. Global numeric variables are initialized to 0. Global characters are initialized to NULL. Global variables have an infinite scope beginning when the variable is 1st defined. They can be accessed by all the f(X)s within its scope like local variables. You should use *global constant*s instead of global variables.

*What's the difference between a global character variable and a numeric global variable?*

// This program has an uninitialized global variable.  
#include <iostream>  
using namespace std;

int globalNum; // Global variable, automatically set to zero

int main()  
{  
 cout << "globalNum is " << globalNum << endl;  
 return 0;  
}

*What will be displayed when above program is executed?*

---

You can't give a parameter variable & a local variable in the same f(x) the same name.

---

To make a local variable have to essentially have the same scope, lifetime, and default initialization as a global variable make the variable static.

EX: static int statNum;

// This program uses a static local variable.  
#include <iostream>  
using namespace std;

void showStatic(); // Function prototype

int main()  
{  
 // Call the showStatic function five times.  
 for (int count = 0; count < 5; count++)  
 showStatic();  
 return 0;  
}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
// Definition of function showStatic. \*  
// statNum is a static local variable. Its value is displayed \*  
// and then incremented just before the function returns. \*  
//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void showStatic()  
{  
 static int statNum;

cout << "statNum is " << statNum << endl;  
 statNum++;  
}

*What will the above program display?*

**Default arguments are passed to parameters automatically if no argument is provided in the function call.**

<https://www.thenewboston.com/videos.php?cat=16&video=17504> skip to 2 min of video

All the default arguments may be overridden. If a f(x) doesn't have a prototype, default arguments may be specified on the f(x) header. Default arguments can't hold parameters. They can hold literals and constants

// This program demonstrates default function arguments.  
#include <iostream>  
using namespace std;

// Function prototype with default arguments  
void displayStars(int = 10, int = 1);

int main()  
{  
 displayStars(); // Use default values for cols and rows.  
 cout << endl;  
 displayStars(5); // Use default value for rows.  
 cout << endl;  
 displayStars(7, 3); // Use 7 for cols and 3 for rows.  
 return 0;  
}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
// Definition of function displayStars. \*  
// The default argument for cols is 10 and for rows is 1.\*  
// This function displays a square made of asterisks. \*  
//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void displayStars(int cols, int rows)  
{  
 // Nested loop. The outer loop controls the rows  
 // and the inner loop controls the columns.  
 for (int down = 0; down < rows; down++)  
 {  
 for (int across = 0; across < cols; across++)  
 cout << "\*";  
 cout << endl;  
 }  
}

**When used as parameters, reference variables allow a f(x) to access the parameter's original argument. Changes to the parameter are also made to the argument.**

**When you pass by reference the function has access to the variables passed to it. When the function changes an argument that was passed by reference it changes the value of the original variable passed to the function.**

From <[*https://elearning.utdallas.edu/bbcswebdav/pid-894609-dt-content-rid-7979608\_1/users/dgv130030/CS%201136%20F15/Lesson%209/CS%201136%20Laboratory%20Lesson%209.pdf*](https://elearning.utdallas.edu/bbcswebdav/pid-894609-dt-content-rid-7979608_1/users/dgv130030/CS%201136%20F15/Lesson%209/CS%201136%20Laboratory%20Lesson%209.pdf)>

**&&&&&&&&&&**

By using a reference variable as a parameter, a function may change a variable that is defined in another function. Reference variables are defined like regular variables, except you place an ampersand (&) in front of the name.

EX:

Void doubleNum( int &refVar )

{

redVar \*= 2;

}

The ampersand must appear in both the prototype & the header of any f(x) that uses a reference variable as a parameter. It doesn't appear in the function call.

// This program uses a reference variable as a function  
// parameter.  
#include <iostream>  
using namespace std;

// Function prototype. The parameter is a reference variable.  
void doubleNum(int &);

int main()  
{  
 int value = 4;

cout << "In main, value is " << value << endl;  
 cout << "Now calling doubleNum..." << endl;  
 doubleNum(value);  
 cout << "Now back in main. value is " << value << endl;  
 return 0;  
}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
// Definition of doubleNum. \*  
// The parameter refVar is a reference variable. The value \*  
// in refVar is doubled. \*  
//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void doubleNum (int &refVar)  
{  
 refVar \*= 2;  
}

When a reference parameter is used, it's said that the argument is *passed by reference*

**Several f(x)s may have the same name, as long as their parameter lists are different**

The compiler will determine which version of function to call *based* upon the *number* and

*data types of the arguments* in the function call.

From <[*https://elearning.utdallas.edu/bbcswebdav/pid-950869-dt-content-rid-9069273\_1/users/lthomp/CS%201336%20Fall%202015/Slides/CS\_1336\_Chapter\_06\_PDF.pdf*](https://elearning.utdallas.edu/bbcswebdav/pid-950869-dt-content-rid-9069273_1/users/lthomp/CS%201336%20Fall%202015/Slides/CS_1336_Chapter_06_PDF.pdf)>

The *function signature* is the name of the f(x) & the data types of the f(x)'s parameters in the proper order. EX:

<https://www.thenewboston.com/videos.php?cat=16&video=17506>

The name of parameters in f(x)s can be overridden by the names of them in their f(x) prototypes?

// This program uses overloaded functions.  
#include <iostream>  
#include <iomanip>  
using namespace std;

// Function prototypes  
int square(int);  
double square(double);

int main()  
{  
 int userInt;  
 double userFloat;

// Get an int and a double.  
 cout << fixed << showpoint << setprecision(2);  
 cout << "Enter an integer and a floating-point value: ";  
 cin >> userInt >> userFloat;  
   
 // Display their squares.  
 cout << "Here are their squares: ";  
 cout << square(userInt) << " and " << square(userFloat);  
 return 0;  
}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
// Definition of overloaded function square. \*  
// This function uses an int parameter, number. It returns the \*  
// square of number as an int. \*  
//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

int square(int number)  
{  
 return number \* number;  
}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
// Definition of overloaded function square. \*  
// This function uses a double parameter, number. It returns \*  
// the square of number as a double. \*  
//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

double square(double number)  
{  
 return number \* number;  
}

**The exit () function causes a program to terminate, regardless of which function or control mechanism is executing.**

A C++ program stops executing when the return statement in f(x) main is encountered. This is not true for other f(x)s. Only when the exit () is used can a program end w/o int main ending.

// This program shows how the exit function causes a program  
// to stop executing.  
#include <iostream>  
#include <cstdlib> // For exit  
using namespace std;

void function(); // Function prototype

int main()  
{  
 function();  
 return 0;  
}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
// This function simply demonstrates that exit can be used \*  
// to terminate a program from a function other than main. \*  
//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void function()  
{  
 cout << "This program terminates with the exit function.\n";  
 cout << "Bye!\n";  
 exit(0);  
 cout << "This message will never be displayed\n";  
 cout << "because the program has already terminated.\n";  
}

EXIT\_SUCCESS can be placed inside the parenthesis of exit() instead of 0. The exit () will work the same with either of those arguments in it.

EXIT\_FAILURE is defined as the termination code that commonly represents an unsuccessful exit under the current operating system. This is used as an argument of exit()

A *stub* is a dummy f(x) that is called instead of the actual f(x) it represents. It usually displays a test message acknowledging that it was called.

A stub allows you to determine whether your program is calling a f(x) when you expect it to, & to confirm that valid values are being sent to the f(x). If the stub reprents a f(x) that returns a value, than the stub should return a test value.

A *driver* is a program that tests a f(x) by simply calling it. If the f(x) accepts arguments, the driver passes test data.

// This program is a driver for testing the showFees function.  
#include <iostream>  
using namespace std;

// Prototype  
void showFees(double, int);

int main()  
{  
 // Constants for membership rates  
 const double ADULT = 40.0;  
 const double SENIOR = 30.0;  
 const double CHILD = 20.0;

// Perform a test for adult membership.  
 cout << "Testing an adult membership...\n"  
 << "Calling the showFees function with arguments "  
 << ADULT << " and 10.\n";  
 showFees(ADULT, 10);  
   
 // Perform a test for senior citizen membership.  
 cout << "\nTesting a senior citizen membership...\n"  
 << "Calling the showFees function with arguments "  
 << SENIOR << " and 10.\n";  
 showFees(SENIOR, 10);  
   
 // Perform a test for child membership.  
 cout << "\nTesting a child membership...\n"  
 << "\nCalling the showFees function with arguments "  
 << CHILD << " and 10.\n";  
 showFees(CHILD, 10);  
 return 0;  
}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
// Definition of function showFees. The memberRate parameter \*  
// the monthly membership rate and the months parameter holds the \*  
// number of months. The function displays the total charges. \*  
//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void showFees(double memberRate, int months)  
{  
 cout << "The total charges are $"  
 << (memberRate \* months) << endl;  
}