2.5 — Why functions are useful, and how to use them effectively

BY ALEX ON AUGUST 4TH, 2015 | LAST MODIFIED BY ALEX ON JUNE 13TH, 2019

Now that we've covered what functions are and some of their basic capabilities, let's take a closer look at why they're useful.

Why use functions?

New programmers often ask, "Can't we just put all the code inside the *main* function?" For simple programs, you absolutely can. However, functions provide a number of benefits that make them extremely useful in programs of non-trivial length or complexity.

- Organization -- As programs grow in complexity, having all the code live inside the main() function becomes increasingly complicated. A function is almost like a mini-program that we can write separately from the main program, without having to think about the rest of the program while we write it. This allows us to reduce a complicated program into smaller, more manageable chunks, which reduces the overall complexity of our program.
- Reusability -- Once a function is written, it can be called multiple times from within the program. This avoids duplicated code ("Don't Repeat Yourself") and minimizes the probability of copy/paste errors. Functions can also be shared with other programs, reducing the amount of code that has to be written from scratch (and retested) each time.
- Testing -- Because functions reduce code redundancy, there's less code to test in the first place. Also because functions are self-contained, once we've tested a function to ensure it works, we don't need to test it again unless we change it. This reduces the amount of code we have to test at one time, making it much easier to find bugs (or avoid them in the first place).
- Extensibility -- When we need to extend our program to handle a case it didn't handle before, functions allow us to make the change in one place and have that change take effect every time the function is called.
- Abstraction -- In order to use a function, you only need to know its name, inputs, outputs, and where it lives. You don't need to know how it works, or what other code it's dependent upon to use it. This lowers the amount of knowledge required to use other people's code (including everything in the standard library).

Although it doesn't look like it, every time you use operator<< or operator>> to do input or output, you're using a function provided by the standard library that meets all of the above criteria.

Effectively using functions

One of the biggest challenges new programmers encounter (besides learning the language) is understanding when and how to use functions effectively. Here are a few basic guidelines for writing functions:

- Statements that appear more than once in a program should generally be made into a function. For example, if we're reading input from the user multiple times in the same way, that's a great candidate for a function. If we output something in the same way multiple times, that's also a great candidate for a function.
- Code that has a well-defined set of inputs and outputs is a good candidate for a function, particularly if it is complicated. For example, if we have a list of items that we want to sort, the code to do the sorting would make a great function, even if it's only done once. The input is the unsorted list, and the output is the sorted list.
- A function should generally perform one (and only one) task.
- When a function becomes too long, too complicated, or hard to understand, it can be split into multiple sub-functions. This is called **refactoring**. We talk more about refactoring in lesson **3.10 -- Finding issues before they become problems**.

Typically, when learning C++, you will write a lot of programs that involve 3 subtasks:

1. Reading inputs from the user
2. Calculating a value from the inputs
3. Printing the calculated value

For trivial programs (e.g. less than 20 lines of code), some or all of these can be done in function *main*. However, for longer programs (or just for practice) each of these is a good candidate for an individual function.

New programmers often combine calculating a value and printing the calculated value into a single function. However, this violates the "one task" rule of thumb for functions. A function that calculates a value should return the value to the caller and let the caller decide what to do with the calculated value (such as call another function to print the value).

 **2.6 -- Whitespace and basic formatting**

 **Index**

 **2.4 -- Introduction to local scope**

C++ TUTORIAL | PRINT THIS POST

56 comments to 2.5 — Why functions are useful, and how to use them effectively

Jonas
June 9, 2019 at 12:58 am · Reply

Found a typo at the first point of 'Effectively using functions' (incorrect word is marked with asterisks and already corrected):

Statements that *appear* more than once in a program should generally be made into a function.

Rowan
February 28, 2019 at 12:16 pm · Reply

do you know why 0 does not move

```
#include <iostream>
#include <conio.h>

using namespace std;
bool gameover;
const int width = 20;
const int height = 20;
int x, y, fruitX,fruitY, score;
enum eDirection { STOP = 0, LEFT, RIGHT, UP, DOWN};
eDirection dir;
void Setup() {
   gameover = false;
   dir = STOP;
   x = width / 2;
   y = height / 2;
   fruitX = rand() % width;
   fruitY = rand() % height;
   score = 0;
}
void Draw() {

   system("cls");
   for (int i = 0; i > width; i++)
      cout << "#";
   cout << endl;

   for (int i = 0; i < height; i++)
   {

      for (int j = 0; j < width; j++)
      {

         if (j == 0)

            cout << "#";
         if (i == y && j == x)
            cout << "0";

         else if (i == fruitY && j == fruitX)
            cout << "F";

         else




            cout << " ";
         if (j == width + 2)
            cout << "#";
      }
      cout << endl;
   }

   for (int i = 0; i > width+2; i++)
      cout << "#";
```

```
    cout << endl;




}
void input() {

    if (_kbhit())
    {
        switch (_getch()) {
        case 'a':
            dir = LEFT;
            break;
        case 'd':
            dir = RIGHT;
            break;
        case 'w':
            dir = UP;
            break;
        case 's':
            dir = DOWN;
            break;
        case 'x':
            gameover = true;
            break;
        }
    }
}
void Logic()
{
    switch (eDirection())
    {
    case STOP:
        break;
    case LEFT:
        break;
    case RIGHT:
        break;
    case UP:
        break;
    case DOWN:
        break;
    default:
        break;
    }
    if (x > width || x < 0 || y > height || y < 0)
        gameover = true;
}

int main()
{
    Setup();
    while (!gameover)
        Draw();
```

```
    input();
    Logic();

    return 0;
}
```

Julius
March 31, 2019 at 7:28 am · Reply

I did not read in detail through all of it but one problem I found was that the GameLoop in Main (while(!gameover))
only calls your draw function, not the input and Logic functions. They only get called once the gameLoop is over.
I would suggest moving the input and Logic funtion calls into the GameLoop and see what happens (change this)

```
while(!gameover) {
    Draw();
}
input();
Logic();
```

to this

```
while(!gameover) {
    input();
    Logic();
    Draw();
}
```

Anonymous
November 8, 2018 at 5:11 am · Reply

I combined all the things I learnt so far up to this point
Is there anything I could improve on in this code or so far so good?

```
#include <iostream>

void doNothing(const int &q)
{
}

int getValueFromUser()
{
    std::cout << "Enter a number: " << std::endl;
    int a;
    std::cin >> a;
    return a;
}

void doRendering()
{
    std::cout << "Rendering..." << std::endl;
}

void doConfiguring()
{
    std::cout << "Configuring..." << std::endl;
}
```

```
void doRenderAndConfigure()
{
   doRendering();
   doConfiguring();
}

void printValue(int x, int y)
{
   std::cout << x << std::endl;
   std::cout << y << std::endl;
}

int add(int x, int y)
{
   return x + y;
}

int multiply(int x, int y)
{
   return x * y;
}

int doubleNumber(int x)
{
   return x * 2;
}

int Return5()
{
   return 5;
}

int main()
{
   int x = getValueFromUser();
   int y = getValueFromUser();
   int z;
   doNothing(z);
   doRenderAndConfigure();
   std::cout << "Hello World!\nYour numbers are: " << add(doubleNumber(x), y) << std::endl;
   std::cout << z << std::endl;
   printValue(Return5(), multiply(Return5(), 2));
   return 0;

}
```

nascardriver
November 8, 2018 at 7:46 am · Reply

* "q" is a poor variable name. Avoid abbreviations unless they're obvious.

Other than that there's not much to say, because this is lesson 1.4b. You'll learn about other improvements in future lessons.
When you're done with lesson 2.7, there should two improvements you can make to your code. If you don't find them, feel free to repost your code or leave a reply here.

Dhruv Tyagi

April 19, 2018 at 12:39 pm · Reply

Dear Alex, I made a small calculator to add, subtract, multiply and devidetwo integers...Can you edit the code such that it can calculate the decimals too?

```
#include "stdafx.h"
#include <iostream>

int add(int x, int y) // add two numbers
{
    return x + y;
}

int multiply(int x, int y) // multiply two numbers
{
    return x * y;
}

int subtract(int x, int y) // subtract two numbers
{
    return x - y;
}

int devide(int x, int y) // devide two numbers
{
    return x / y;
}

int req1()  //choose the first number
{
    int a;
    std::cout << "Enter the first variable: " << std::endl;
    std::cin >> a;
    return a;
}

int req2()   //choose the second number
{
    int b;
    std::cout << "Enter the second variable: " << std::endl;
    std::cin >> b;
    return b;
}

int req3() //choose the operator
{
    int op;
    std::cout << "Enter the mathematical operator: (1 = + , 2 = - , 3 = * , 4 = /) " << std
    std::cin >> op;
    return op;
}

void calculate() {   //calculate
    int p, q, r;
    p = req1();  // number 1
    q = req2();  //number 2
    r = req3();  // operator
    if (r == 1) {  // add
        std::cout << p << " + " << q << " = " << add(q, p) << std::endl;
    }
    else if (r == 2) {   // subtract
        std::cout << p << " - " << q << " = " << subtract(p, q) << std::endl;
```

```
    }
    else if (r == 3) {  //multiply
        std::cout << p << " * " << q << " = " << multiply(p, q) << std::endl;
    }
    else if (r == 4) {  // devide
        std::cout << p << " / " << q << " = " << devide(p, q) << std::endl;
    }
    else {  // invalid operator
        std::cout << "Enter a valid operator..." << std::endl;
    }
}
int main() {
    calculate();
}
```

Thanks

nascardriver
April 20, 2018 at 8:36 am · Reply

Hi Dhruv!

Use 'double' instead of 'int'.

References
Lesson 2.1 - Fundamental variable definition, initialization, and assignment
Lesson 2.5 - Floating point numbers

Singh
June 18, 2019 at 10:08 am · Reply

Hi there, How's it going?

Can you please look into my case. Please check with line number 11 and 56.

1. Unable to use double to get decimal value on a division of number.
2. Not able to clear screen if not a valid input from a user.

```
//Program is to use different functions to get the results.

#include <iostream>

int addfunction(int a, int b)
    { return a+b; }

int multi(int a, int b)
    { return a*b; }

int division(int a, int b) //not getting result even using double.
    { double c=a/b;
        std::cout<<"Division of both A & B is: " <<c;
        return c; }

int main()
    {
        std::cout<<"Program is to add, mulitply or divide the numbers.\n";
        int x{};
        std::cout<<"Please select the function.\nTo ADD: 1\nTo MUL: 2\nTo DIV: 3\
        std::cin>>x;

```

```
        // And how can I use the below set of code in new function which can be c
        //only after condition check whereas it has to run every time. But its wo
        //"else if(x>=4)" condition as "if(x>=4)" below (before a & b declaration
        //    if(x>=4)
        //      {
        //         std::cout<<"Please input valid number.\n";
        //         std::cin.clear();
        //         main();
        //      }
        //    Is it a good approach or do we have any solution to it. */

        int a, b;
        std::cout<<"\nPlease enter the numbers.";
        std::cout<<"A: ";
        std::cin>>a;
        std::cout<<"B: ";
        std::cin>>b;

        if(x==1)
        {
            std::cout<<addfunction(a,b);
        }
            else if(x==2)
            {
                std::cout<<multi(a,b);
            }
             else if(x==3)
             {
                 std::cout<<division(a,b);
             }
                else if(x>=4)
                {
                    std::cout<<"Please input valid number.\n";
                    system("cls"); //how to perform clear screan and again it goe
                    main();
                }

    return 0;
}
```

nascardriver
June 19, 2019 at 3:24 am · Reply

- Use your editor's auto-formatting feature.
- `main` cannot be used. You need a loop, loops are covered later.
- Don't use `system`, it won't work on other platforms.

> Line 12
`c`'s type doesn't matter. You're dividing an `int` by an `int`, the result is an `int`, which is then converted to a `double`. You need to copy `a` or `b` or both into a double and divide that. (You'll learn about casts later, you don't need an extra variable then).

> Line 56
There is no universal way to clear the console/terminal. You can try hiding the previous output by printing a lot of line feeds.

```
std::cout << std::string(1024, '\n'); // Repeats '\n' 1024 times.
// I made up 1024. There is no universal way of retrieving the terminal height
```

Henry
April 4, 2018 at 7:38 am · Reply

Hi Alex,please what does these three statements do:

"std::cin.clear();
std::cin.ignore(32767, '\n');
std::cin.get();" ????

nascardriver
April 4, 2018 at 7:46 am · Reply

Hi Henry!

This is covered in lesson 5.10.

@std::cin.clear clears @std::cin's error flag
@std::cin.ignore ignores all characters in the input stream until '\n' is found, but a maximum of 32767 characters.
@std::cin.get reads one character from the input stream

Henry
April 4, 2018 at 11:46 pm · Reply

Thank you for this tutorial.
I used to be scared of programming before now, but this tutorial has changed that.

Anonymous
April 20, 2018 at 8:04 am · Reply

you can use double instead of int to define all the variables and you will be able to do decimal calculations as well.

ScarLet
July 6, 2017 at 8:49 pm · Reply

Is in a program only contain one function main() or can be more in a program?

Alex
July 6, 2017 at 11:19 pm · Reply

A program can only contain one main() function, but may contain as many other functions as you want.

Kamran
July 6, 2017 at 3:38 am · Reply

I want to learn c++

UncleMi
July 7, 2017 at 8:35 am · Reply

You are in the right place then

hasanen
February 1, 2017 at 11:46 am · Reply

#include<iostream.h>
void main()
{
for(int i=1;i<20;i++)
{for(int s=20;s>i;s--)
{cout<<" ";}
for(int j=1;j<=i;j++)
{cout<<" $";}
for(int g=1;g<i;g++)
{cout<<"$";}
cout<<endl;}
}

WiseFool
February 26, 2018 at 12:34 pm · Reply

cool little program, hasanen, but I had to debug it to get it to run:

1) my compiler insisted that main() return int rather than void.
2) I had to put "using namespace std;" after the #include to get
it to compile right on my compiler as written.
3) the s-minus on line 5 should be s-minus-minus.
(I noticed it wouldn't let me put 2 minuses after it
either, so it may be the comment editor rather
than your error.)

Hoin&CoinYT
February 26, 2018 at 3:27 pm · Reply

do you know where to put in the program?

Kanishka Williamson
January 23, 2017 at 10:03 pm · Reply

Hi Alex,

You are impressive. I want to learn C++ at the age of 12 and i am of 12 now. This make a mastered in C++ programing.

Thanks Alex! Thanks very very very much!

Anonymous Account
April 21, 2017 at 9:31 am · Reply

Hey Kanishka! I'm 15 and I started learning at the age of 7, this tutorial is awesome for learning C++, if this is the first programming language you learn and you find it interesting,

congratulations! Nowadays it's an useful tool to understanding how everything works. When I join a website and I see the design I think: "this is created using margin-left at a 15% or using a container with a col of 2 at both sides".

I just wanted to tell you good luck with your development adventure :)

Xavier
July 5, 2017 at 1:07 pm · Reply

wow, way to make me feel like ive been wasting my life away XD (17 yrs old)
ps why would you think in bootstrap??

Charlie
March 24, 2018 at 8:46 pm · Reply

I am 32 and just started learning it. It is never too late.

SHOULIE
March 28, 2018 at 9:47 am · Reply

Am 14 , I always miss the ;!!!!!!!
; is used for suicide awareness ,
And am sure this silly mistake will frustrate people and make them do it....

GEScott71
December 27, 2016 at 7:01 am · Reply

Thanks Alex, this is a great explanation of functions. I really appreciate you explaining the "why" behind many aspects of C++.

ramin
December 1, 2016 at 8:45 am · Reply

tanx alex

hakeem
October 10, 2016 at 4:30 pm · Reply

I can't believe how awesome this tutorial is!!! I wonder if Alex has one for GoLang because that's next on my list, anyways, kudos!!!

reznov
September 11, 2016 at 10:06 am · Reply

So basicly instead of using namespaces you could also just write a function that uses std::cout and call a function that's defined as printNumber(int x) for integers.
My question is would this also work if I want to print text this way? Defining my function as printText(string x) ???

Alex

September 12, 2016 at 1:01 pm · Reply

Yes, if you use std::string. We cover the basics of std::string in a future lesson.

SploingOne

June 25, 2016 at 10:48 pm · Reply

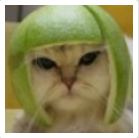
Hey Alex, I made this simple calculator program after last tutorial and I am wondering whether I should have split calculations and things up into functions, what do you think?

```
#include "stdafx.h"
#include <iostream>

int main()
{
    bool play = true;

    std::cout << "This is a simple two integer equation calculator" << std::endl;
    std::cout << "================================================" << std::endl;
    std::cout << "-This calculator takes only real numbers" << std::endl;
    std::cout << "-This calculator can handle + - / * ^ < > =" << std::endl;
    std::cout << "-Output is rounded to nearest whole number" << std::endl;
    std::cout << "================================================" << std::endl;
    while (play)
    {
        std::cout << "Enter your first number: ";
        int num1;
        std::cin >> num1;
        std::cout << "Enter your second number: ";
        int num2;
        std::cin >> num2;
        std::cout << "Enter your operator: ";
        char op;
        std::cin >> op;
        switch (op)
        {
        case '+':
            std::cout << num1 + num1 << std::endl;
            break;
        case '-':
            std::cout << num1 - num2 << std::endl;
            break;
        case '/':
            std::cout << num1 / num2 << std::endl;
            break;
        case '*':
            std::cout << num1 * num2 << std::endl;
            break;
        case '^':
            std::cout << std::pow(num1, num2) << std::endl;
            break;
        case '<':
            if (num1 < num2)
            {
                std::cout << "true" << std::endl;
            }
            else
            {
                std::cout << "false" << std::endl;
            }
```

```
            break;
        case '>':
            if (num1 > num2)
            {
                std::cout << "true" << std::endl;
            }
            else
            {
                std::cout << "false" << std::endl;
            }
            break;
        case '=':
            if (num1 == num2)
            {
                std::cout << "true" << std::endl;
            }
            else
            {
                std::cout << "false" << std::endl;
            }
            break;
        default:
            std::cout << "invalid operator" << std::endl;
            break;
        }
        std::cout << "Would you like to make another calculation?" << std::endl;
        char ans;
        std::cin >> ans;
        switch (ans)
        {
        case 'n':
            play = false;
            break;
        case 'N':
            play = false;
            break;
        case 'y':
            play = true;
            break;
        case 'Y':
            play = true;
            break;
        default:
            std::cout << "invalid response - defaulting to no";
            play = true;
            break;
        }
    }
    return 0;
}
```



Alex
June 26, 2016 at 11:24 am · Reply

Yes, this program is pretty long for one function. It would definitely benefit from being split into smaller functions.

African
October 23, 2017 at 7:32 am · Reply

what lesson do u learn this from???

Darren
June 2, 2016 at 8:48 am · Reply

Learning the functionality of functions, in other words how functions function, was the main function of this page about functions. Function.

AmnesiaMaster28
April 30, 2016 at 2:41 am · Reply

Hey Alex, I'm curious. Are there any guidelines to doing anything inside main? Like, is it better to do:

```
#include "stdafx.h"
#include <iostream>

int enterNumber()
{
    int x;
    std::cout << "Enter a number: ";
    std::cin >> x;
    return x;
}

int main()
{
    int x{enterNumber()};
    int y{enterNumber()};
    int z{x + y};
    std::cout << x << " plus " << y << " equals " << z;
    std::cin.clear();
    std::cin.ignore(32767, '\n');
    std::cin.get();
    return 0;
}
```

Or

```
#include "stdafx.h"
#include <iostream>

int enterNumber()
{
    int x;
    std::cout << "Enter a number: ";
    std::cin >> x;
    return x;
}

void result(int x, int y)
{
    int z{ x + y };
    std::cout << x << " plus " << y << " equals " << z;
}

int main()
```

```
{
    int x{ enterNumber() };
    int y{ enterNumber() };
    int z{ x + y };
    std::cout << x << " plus " << y << " equals " << z;
    std::cin.clear();
    std::cin.ignore(32767, '\n');
    std::cin.get();
    return 0;
}
```

Alex
April 30, 2016 at 12:25 pm · Reply

Personally I like to keep my main() functions as simple as possible and delegate responsibilities to subfunctions, but that's just a matter of style. When programs are this short, it doesn't matter all that much.

Khoa Vu
June 29, 2018 at 6:56 am · Reply

Hello AmnesiaMaster28,
I have revised your calculator a little bit if you want to take a look. Thanks!

```
#include <cstdlib>
#include <iostream>
#include <cmath>
#include <limits>

using namespace std;

/*
*
*/

double getValueFromUser()
{
   double a;
   std::cin >> a;
   return a;
}

char getValueFromUser2(){
   char b;
   std::cin >> b;
   return b;
}

void Comparison(char z, int x, int y){

   if(z == '+'){ std::cout << x + y << std::endl; }
   else if(z == '-'){ std::cout << x - y << std::endl; }
   else if(z == '*'){ std::cout << x * y << std::endl; }
   else if(z == '/'){ float w = ((float)x/y); printf("%f", w); std::cout << std::endl; }
   else if(z == '^'){ std::cout << std::pow(x,y) << std::endl; }
   else if(z == '<'){
      if(x < y){ std::cout << "True" << std::endl; }
      else{ std::cout << "False" << std::endl; }
```

```
    }
    else if(z == '>'){
        if(x > y){ std::cout << "True" << std::endl; }
        else{ std::cout << "False" << std::endl; }
    }
    else if(z == '='){
        if(x == y){ std::cout << "True" << std::endl; }
        else{ std::cout << "False" << std::endl; }
    }
    else{ std::cout << "Invalid Operator" << std::endl; }
    //return 0;
}

int main(int argc, char** argv) {
    bool play = true;

    std::cout << "This is a simple two-real-number-equation calculator" << std::endl;
    std::cout << "=================================================" << std::endl;
    std::cout << "-This calculator takes only real numbers" << std::endl;
    std::cout << "-This calculator can handle + - / * ^ < > =" << std::endl;
    std::cout << "-Output is rounded to nearest whole number" << std::endl;
    std::cout << "=================================================" << std::endl;
    while (play == true)
    {
        std::cout << "Enter your first number: ";
        double num1 = getValueFromUser();
        std::cout << "Enter your second number: ";
        double num2 = getValueFromUser();
        std::cout << "Enter your operator: ";
        char op = getValueFromUser2();

        Comparison(op, num1, num2);

        string s;
        std::cout << "Would you like to make another calculation?" << std::endl;
        std::cin >> s;
        if(s == "Y" || s == "y" || s == "yes" || s == "Yes" || s == "YES"){ play = true; }
        else if(s == "N" || s == "n" || s == "no" || s == "No" || s == "NO"){ play = false; }
        else{  play = false; std::cout << "invalid response - defaulting to no" << std::endl; }
    }
    return 0;
}
```

rdx
March 20, 2016 at 3:20 am · Reply

hey alex,,,,hats off,,,a great tutorial,,

Tony
March 6, 2016 at 9:40 am · Reply

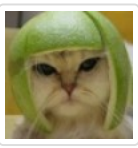
I would say easily one of the best tutorials so far. I've been through many tutorials that overlook certain points and don't go into detail of what certain things mean... example "using namespace std", many tutorials do not explain what this is and why it's needed of in your case, needed at all.

Kudos to you, and can't wait to go through the rest of the pages.

Tahir
February 23, 2016 at 2:04 pm · Reply

every time you use std::cin or std::cout to do input or output, you’re using a function provided by the standard library that meets all of the above criteria.

This statement is not correct ...
cin and cout are not functions they are objects and when we use them the operator overloaded function cout.operator<<(expr) will be called.

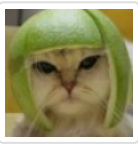


Alex
February 28, 2016 at 12:30 pm · Reply

The statement is correct as written. As you note yourself, when we use cin or cout to do input and output (via operator<&lt or operator>&gt), an overloaded function is called. That function meets the criteria listed.

Jim
February 14, 2016 at 10:59 pm · Reply

Alex,
Thanks for adding this section. However it brings up a lot of question to mind. How do the pro's store the functions they write? Do they store functions for say integers and floats in different places.? You mentioned retesting, does this mean that they don't recompile the function again? Can you give us some idea of how to do all these and expand on some of the points you listed above. I would think a set of functions to add, subtract, multiply and divide two variables could be used for all of the intake,long,long long, float, etc., could be written with the same code with minor changes..how would the pro's do that. You could write a book on the things you mentioned in this section alone.
Thanks



Alex
February 15, 2016 at 12:07 pm · Reply

Generally, pros will start creating pairs of header (.h) and code (.cpp) files that contain one of two things:
1) A set of related functions -- For example, math.h and math.cpp would contain math-related functions (including versions for both int and double, if that's relevant)
2) A new data type -- For example, if you're doing geometry, you might need a way to represent a Cartesian point. You could create a point.h and point.cpp that contains the definition for this new type.

Once these files have been created and tested once, they can be copied and reused in other programs without having to retest them.

You're correct that functions to do arithmetic on different types (e.g. int and double) are essentially identical -- and yet, C++ makes you declare them as separate functions. This is where template programming comes in. You can declare a template function, and C++ will instantiate different flavors of that function for each type you call it with. So you could write a single template for function add(), and C++ could create an int and a double version based on this. This can be useful with functions -- it's even more useful with "container objects" like arrays that need to be able to hold different types. Since template programming is an advanced concept (and the syntax is headache inducing), we talk about it much later in this tutorial series.

Skizzy
January 16, 2016 at 8:20 pm · Reply

Hey there Alex just want to say i appreciate this site alot so far. I like how much detail and background goes into each lesson. Much better than most prior tuts ive read. in fact the program from the last quiz we had to do, i made very easily.
Thanks!

cei
January 9, 2016 at 4:05 pm · Reply

Hi. Thank you for this site, it is really helpful :) .
I would like to point out that there is a typo at the Reusability paragraph: it should be: "Once a function is written, it can *be* called [...]".

Alex
January 10, 2016 at 5:19 pm · Reply

Fixed, thanks!

Miroslav
November 28, 2015 at 8:36 am · Reply

Hello Alex, I really like your tutorial so far. I find it the easiest to learn from all of which I tried in the internet. Please keep up the good work :)
Also I've got one question. You call a function which uses an argument and then returns a value that you will use later on. In the function though you change the argument that is passed to it. Is there a way to use that new value in main() (Basically I tried returning 2 values in one function xD of course it didn't work, but I was wondering how I can use an argument from a function in main())

Alex
November 28, 2015 at 1:26 pm · Reply

Yes, in chapter 7 we talk about reference parameters, which allow the value of a parameter that is changed in a function to be used by the caller.

Here's a sneak peak:

```
1  void change(int &x) // the & makes x a reference to the argument passed in rather thai
2  {
3      x = 5;
4  }
5
6  int main()
7  {
8      int x = 1;
9      change(x);
10     std::cout << x;
11
12     return 0;
13 }
```

jamesL

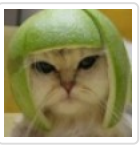
May 1, 2017 at 9:09 am · Reply

So basically you pass a pointer to the variable in the function :) This got me thinking, if you can access directly the address of a variable is it possible to change the address of that variable? I tried this code, but it is giving me an error:

```
1  #include <iostream>
2
3  int main()
4  {
5      const int INTLEN = 4;
6      int x = 0;
7      std::cout << &x << std::endl;
8      std::cout << x << std::endl;
9      &x = &x + INTLEN;
10     std::cout << &x << std::endl;
11     std::cout << x << std::endl;
12
13     return 0;
14 }
```

First it prints out the address of x and its value, then it is supposed to move it up in memory by 4 bytes (size of a 32 bit signed int). BUT line 9 (line where I try to change the address) gives an error, "l-value required as left operand of assignment". So I guess that means you cant change the address like that because it isn't a valid way :(

edit: or i guess it could be that variable addresses are read-only, I guess that could be a valid explanation as well.



Alex

May 1, 2017 at 6:22 pm · Reply

You can't change the address of a variable.

However, you can do something similar to what you're suggesting by using pointers. Have a read about pointers in chapter 6 -- particularly around pointer arithmetic.

HitoShura

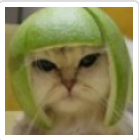
November 14, 2015 at 6:39 pm · Reply

Thanks so much for explaining a lot on this topic and making it easier for us (me) newcomers to understand
much appreciated

Avneet

August 6, 2015 at 7:49 am · Reply

Can a function call itself? Does that make any sense?



Alex

August 6, 2015 at 10:12 am · Reply

Yes, it can. This is called a **recursive** function call. There are quite a few algorithms that are more easily written using recursive function calls.

I talk about recursive function calls in lesson **7.10 -- Recursion**.

Ishana
August 4, 2015 at 9:21 pm · Reply

Hi Alex,

Thank you for putting this new page up. It has given me a better insight to understand the functionality of functions in C++.
And thank you for this great C++ tutorial website:)

Alex
August 5, 2015 at 9:44 am · Reply

You're welcome. Was the article clear? Are there any points that can be clarified further?

Ishana
August 5, 2015 at 8:07 pm · Reply

The article is clear as it is. It clearly communicated the idea of why functions are beneficial in programming. At first, I just learned about functions thinking of it as a necessary requirement, but this article helped me understood its meaning in much deeper context.