2.13 — How to design your first programs

BY ALEX ON SEPTEMBER 5TH, 2009 | LAST MODIFIED BY ALEX ON APRIL 23RD, 2019

Now that you've learned some basics about programs, let's look more closely at *how* to design a program.

When you sit down to write a program, generally you have some kind of idea for which you'd like to write a program for. New programmers often have trouble figuring out how to convert that idea into actual code. But it turns out, you have many of the problem solving skills you need already, acquired from everyday life.

The most important thing to remember (and hardest thing to do) is to design your program *before you start coding*. In many regards, programming is like architecture. What would happen if you tried to build a house without following an architectural plan? Odds are, unless you were very talented, you'd end up with a house that had a lot of problems: walls that weren't straight, a leaky roof, etc... Similarly, if you try to program before you have a good game-plan moving forward, you'll likely find that your code has a lot of problems, and you'll have to spend a lot of time fixing problems that could have been avoided altogether with a little thinking ahead.

A little up-front planning will save you both time and frustration in the long run.

In this lesson, we'll lay out a generalized approach for converting ideas into simple functional programs.

Design step 1: Define your goal

In order to write a successful program, you first need to define what your goal is. Ideally, you should be able to state this in a sentence or two. It is often useful to express this as a user-facing outcome. For example:

- Allow the user to organize a list of names and associated phone numbers.
- Generate randomized dungeons that will produce interesting looking caverns.
- Generate a list of stock recommendations.
- Model how long it takes for a ball dropped off a tower to hit the ground.

Although this step seems obvious, it's also highly important. The worst thing you can do is write a program that doesn't actually do what you (or your boss) wanted!

Design step 2: Define requirements

While defining your problem helps you determine *what* outcome you want, it's still vague. The next step is to think about requirements.

Requirements is a fancy word for both the constraints that your solution needs to abide by (e.g. budget, timeline, space, memory, etc...), as well as the capabilities that the program must exhibit in order to meet the users' needs. Note that your requirements should similarly be focused on the "what", not the "how".

For example:

- Phone numbers should be saved, so they can be recalled later.
- The randomized dungeon should always contain a way to get from the entrance to an exit.
- The stock recommendations should leverage historical pricing data.
- The user should be able to enter the height of the tower.
- We need a testable version in 7 days.

A single problem may yield many requirements, and the solution isn't "done" until it satisfies all of them.

Design step 3: Define your tools, targets, and backup plan

When you are an experienced programmer, there are many other steps that typically would take place at this point, including:

- Defining what target architecture and/or OS your program will run on.
- Determining what set of tools you will be using.
- Determining whether you will write your program alone or as part of a team.
- Defining your testing/feedback/release strategy.
- Determining how you will back up your code.

However, as a new programmer, the answers to these questions are typically simple: You are writing a program for your own use, alone, on your own system, using an IDE you purchased or downloaded, and your code is probably not used by anybody but you. This makes things easy.

That said, if you are going to work on anything of non-trivial complexity, you should have a plan to backup your code. It's not enough to just zip or copy the directory to another location on your machine (though this is better than nothing). If your system crashes, you'll lose everything. A good backup strategy involves getting a copy of the code off of your system altogether. There are lots of easy ways to do this: Zip it up and email it to yourself, copy it to Dropbox or another cloud service, FTP it to another machine, copy it to another machine on your local network, or use a version control system residing on another machine or in the cloud (e.g. github). Version control systems have the added advantage of not only being able to restore your files, but also to roll them back to a previous version.

Design step 4: Break hard problems down into easy problems

In real life, we often need to perform tasks that are very complex. Trying to figure out how to do these tasks can be very challenging. In such cases, we often make use of the **top down** method of problem solving. That is, instead of solving a single complex task, we break that task into multiple subtasks, each of which is individually easier to solve. If those subtasks are still too difficult to solve, they can be broken down further. By continuously splitting complex tasks into simpler ones, you can eventually get to a point where each individual task is manageable, if not trivial.

Let's take a look at an example of this. Let's say we want to clean our house. Our task hierarchy currently looks like this:

- Clean the house

Cleaning the entire house is a pretty big task to do in one sitting, so let's break it into subtasks:

- Clean the house
    - Vacuum the carpets
    - Clean the bathrooms
    - Clean the kitchen

That's more manageable, as we now have subtasks that we can focus on individually. However, we can break some of these down even further:

- Clean the house
    - Vacuum the carpets
    - Clean the bathrooms
        - Scrub the toilet (yuck!)
        - Wash the sink
    - Clean the kitchen
        - Clear the countertops
        - Clean the countertops
        - Scrub the sink
        - Take out the trash

Now we have a hierarchy of tasks, none of them particularly hard. By completing each of these relatively manageable sub-items, we can complete the more difficult overall task of cleaning the house.

The other way to create a hierarchy of tasks is to do so from the **bottom up**. In this method, we'll start from a list of easy tasks, and construct the hierarchy by grouping them.

As an example, many people have to go to work or school on weekdays, so let's say we want to solve the problem of "go to work". If you were asked what tasks you did in the morning to get from bed to work, you might come up with the following list:

- Pick out clothes
- Get dressed
- Eat breakfast
- Drive to work
- Brush your teeth
- Get out of bed
- Prepare breakfast
- Get in your car
- Take a shower

Using the bottom up method, we can organize these into a hierarchy of items by looking for ways to group items with similarities together:

- Get from bed to work
    - Bedroom things
        - Get out of bed
        - Pick out clothes
        - Get dressed
    - Bathroom things
        - Take a shower
        - Brush your teeth
    - Breakfast things
        - Prepare cereal
        - Eat cereal
    - Transportation things
        - Get in your car
        - Drive to work

As it turns out, these task hierarchies are extremely useful in programming, because once you have a task hierarchy, you have essentially defined the structure of your overall program. The top level task (in this case, "Clean the house" or "Go to work") becomes main() (because it is the main problem you are trying to solve). The subitems become functions in the program.

If it turns out that one of the items (functions) is too difficult to implement, simply split that item into multiple sub-items/sub-functions. Eventually you should reach a point where each function in your program is trivial to implement.

Design step 5: Figure out the sequence of events

Now that your program has a structure, it's time to determine how to link all the tasks together. The first step is to determine the sequence of events that will be performed. For example, when you get up in the morning, what order do you do the above tasks? It might look like this:

- Bedroom things
- Bathroom things
- Breakfast things
- Transportation things

If we were writing a calculator, we might do things in this order:

- Get first number from user
- Get mathematical operation from user
- Get second number from user
- Calculate result
- Print result

At this point, we're ready for implementation.

Implementation step 1: Outlining your main function

Now we're ready to start implementation. The above sequences can be used to outline your main program. Don't worry about inputs and outputs for the time being.

```
int main()
{
//    doBedroomThings();
//    doBathroomThings();
//    doBreakfastThings();
//    doTransportationThings();

    return 0;
}
```

Or in the case of the calculator:

```
int main()
{
    // Get first number from user
//    getUserInput();

    // Get mathematical operation from user
//    getMathematicalOperation();

    // Get second number from user
//    getUserInput();

    // Calculate result
//    calculateResult();

    // Print result
//    printResult();

    return 0;
}
```

Note that if you're going to use this "outline" method for constructing your programs, your functions won't compile because the definitions don't exist yet. Commenting out the function calls until you're ready to implement the function definitions is one way to address this (and the way we'll show here). Alternatively, you can *stub out* your functions (create placeholder functions with empty bodies) so your program will compile.

Implementation step 2: Implement each function

In this step, for each function, you'll do three things:

1. Define the function prototype (inputs and outputs)
2. Write the function
3. Test the function

If your functions are granular enough, each function should be fairly simple and straightforward. If a given function still seems overly-complex to implement, perhaps it needs to be broken down into subfunctions that can be more easily implemented (or it's possible you did something in the wrong order, and need to revisit your sequencing of events).

Let's do the first function from the calculator example:

```
#include <iostream>

// Full implementation of the getUserInput function
int getUserInput()
{
    std::cout << "Enter an integer ";
    int input{ 0 };
    std::cin >> input;

    return input;
}

int main()
{
    // Get first number from user
    int value = getUserInput(); // Note we've included code here to test the return value!
    std::cout << value;

    // Get mathematical operation from user
//    getMathematicalOperation();

    // Get second number from user
//    getUserInput();

    // Calculate result
//    calculateResult();

    // Print result
//    printResult();

    return 0;
}
```

First, we've determined that the *getUserInput* function takes no arguments, and will return an int value back to the caller. That gets reflected in the function prototype having a return value of int and no parameters. Next, we've written the body of the function, which is a straightforward 4 statements. Finally, we've implemented some temporary code in function *main* to test that function *getUserInput* (including its return value) is working correctly.

We can run this program many times with different input values and make sure that the program is behaving as we expect at this point. If we find something that doesn't work, we know the problem is in the code we've just written. Once we're convinced the program is working as intended up to this point, we can remove the temporary testing code, and proceed to implementation of the next function (function *getMathematicalOperation*).

Remember: Don't implement your entire program in one go. Work on it in steps, testing each step along the way before proceeding.

Implementation step 3: Final testing

Once your program is "finished", the last step is to test the whole program and ensure it works as intended. If it doesn't work, fix it.

Words of advice when writing programs

Keep your programs simple to start. Often new programmers have a grand vision for all the things they want their program to do. “I want to write a role-playing game with graphics and sound and random monsters and dungeons, with a town you can visit to sell the items that you find in the dungeon” If you try to write something too complex to start, you will become overwhelmed and discouraged at your lack of progress. Instead, make your first goal as simple as possible, something that is definitely within your reach. For example, “I want to be able to display a 2-dimensional field on the screen”.

Add features over time. Once you have your simple program working and working well, then you can add features to it. For example, once you can display your field, add a character who can walk around. Once you can walk around, add walls that can impede your progress. Once you have walls, build a simple town out of them. Once you have a town, add merchants. By adding each feature incrementally your program will get progressively more complex without overwhelming you in the process.

Focus on one area at a time. Don’t try to code everything at once, and don’t divide your attention across multiple tasks. Focus on one task at a time. It is much better to have one working task and five that haven’t been started yet than six partially-working tasks. If you split your attention, you are more likely to make mistakes and forget important details.

Test each piece of code as you go. New programmers will often write the entire program in one pass. Then when they compile it for the first time, the compiler reports hundreds of errors. This can not only be intimidating, if your code doesn’t work, it may be hard to figure out why. Instead, write a piece of code, and then compile and test it immediately. If it doesn’t work, you’ll know exactly where the problem is, and it will be easy to fix. Once you are sure that the code works, move to the next piece and repeat. It may take longer to finish writing your code, but when you are done the whole thing should work, and you won’t have to spend twice as long trying to figure out why it doesn’t.

Don’t invest in perfecting early code. The first draft of a feature (or program) is rarely good. Furthermore, programs tend to evolve over time, as you add capabilities and find better ways to structure things. If you invest too early in polishing your code (adding lots of documentation, full compliance with best practices, making optimizations), you risk losing all of that investment when a code change is necessary. Instead, get your features minimally working and then move on. As you gain confidence in your solutions, apply successive layers of polish. Don’t aim for perfect -- non-trivial programs are never perfect, and there’s always something more that could be done to improve them. Get to good enough and move on.

Most new programmers will shortcut many of these steps and suggestions (because it seems like a lot of work and/or it’s not as much fun as writing the code). However, for any non-trivial project, following these steps will definitely save you a lot of time in the long run. A little planning up front saves a lot of debugging at the end.

The good news is that once you become comfortable with all of these concepts, they will start coming more naturally to you. Eventually you will get to the point where you can write entire functions without any pre-planning at all.

 **2.x -- Chapter 2 summary and quiz**

 **Index**

 **2.12 -- Header guards**

C++ TUTORIAL | PRINT THIS POST

326 comments to 2.13 — How to design your first programs

« Older Comments 1 ... 3 4 5

Omran
January 30, 2020 at 7:19 am · Reply

hello this is my program so far , hope you like it and judge me the way you want , :)

Main.cpp

```
#include <iostream>
#include "ForwardDeclaration.h"
#include <cmath>
#define print(x) std::cout<< x ; // i did this to abbreviate std::cout...etc
#define get(x) std::cin >> x; // same with this

int main() {
    while (true) {
        char s;
        print('\n');
        print("note : click 's' if you want to find the square root of a number or click 'b' to start basic operations : \n");
        get(s);
        if (s == 'b') {
            print('\n');
            print("hello , this is a simple calculator\n");
            print('\n');
            float x = GetValueFromUser();
            char  z = Opperator();
            float y = GetValueFromUser();
            if (z == '+' or z == '-' or z == '*' or z == '/') {
                std::cout << x << " " << z << " " << y << " = " << calculator(x, z, y) << '\n';
                print('\n');
                print("your prgram has ran seccessfully !!!!");
                print('\n');
            }
            else {
                print('\n');
                print("you entered invalid symbol , something went wrong , we couldn't complete your operation\n");
                print('\n');
                print("compilation failed!!!!\n")
            }
            print('\n');
        }

        else if(s=='s') {
            print('\n');
            print("enter your rational number here :\n");
            float q;
            get(q);
            print("the square root of " << q << ' = ' << sqrt(q));
            print('\n');
        }
        else {
            print("please click s or b!!!!! \n");
```

```
        }

    }
        return 0;
}
```

MathWork.cpp

```
#include <iostream>
#define print(x) std::cout<< x ; // i did this to abbreviate std::cout...etc
#define get(x) std::cin >> x; // same with this

float GetValueFromUser() {

    print("enter a rational number :\n");
    float input{};
    get(input);
    return input;
}
char Opperator() {
    print("enter a valid operator such as'+' and '-' , '/' , '*' : \n");
    char symbol;
    get(symbol);
    return symbol;
}
float calculator(float term1, char symbol, float term2) {
    if (symbol == '+') {
        print('\n');
        return term1 + term2;
    }
    else if (symbol == '-') {
        print('\n');
        return term1 - term2;
    }
    else if (symbol == '*') {
        print('\n');
        return term1 * term2;
    }
    else if (symbol == '/') {
        print('\n');
        return term1 / term2;
    }
    return 0;
}
```

ForwardDeclaration.h

```
#ifndef header // i was actually going to use #pragma once cuz my compiler support this fea
ture
//but i'll stick around with this
#define header
float GetValueFromUser();
char Opperator();
float calculator(float term1, char symbol, float term2);
#endif
```

nascardriver
January 30, 2020 at 7:55 am · Reply

- Unless you have a reason to use a `float`, use a `double`.
- Use single quotation marks for characters (' ' instead of " ")
- Name variables descriptively. x, y, z, q, s, have no meaning.

- `sqrt` is a C function, use `std::sqrt`.
- `0` is an integer, `0.0f` is a `float`.
- Inconsistent formatting. Use your editor's auto-formatting feature.

The macros aren't good, but you don't know how to do it properly yet.

Omran
January 30, 2020 at 9:08 am · Reply

i'm so sorry i was going to delete it before you see it , i didn't realise that it has a lot of errors , ahh i'm done , i'm so sorry , i forgot about the errors , but i can't delete this comment anymore , why ???

nascardriver
January 30, 2020 at 9:11 am · Reply

Those aren't errors, just simple mistakes that everyone makes on their journey through C++ :)
A working program is a success

Omran
January 30, 2020 at 10:50 pm · Reply

is this fixed version of it okay ?? , can i ask ? , how to use macros ? and am i going to learn how to use them in future lessons ?? , i'd like to hear your answer please , if this isn't good just tell me , " plus i love this websites it's really helpful i was learning c++ from sololearn i didn't understand it properly as i do now i promise to click on every add on this website :) "

Main.cpp

```
1  #include <iostream>
2  #include "ForwardDeclaration.h"
3  #include <cmath>
4
5  int main() {
6      while (true) {
7
8          char Square_Root;
9          std::cout << "\n";
10         std::cout << "note : click 's' if you want to find the square root of a
    number ,or click 'b' to begin basic operations : \n";
11         std::cin >> Square_Root;
12         if (Square_Root== 'b') {
13             std::cout << "\n";
14             std::cout << "hello , this is a simple calculator\n";
15             std::cout << "\n";
16             double GetTerm1FromUser{ GetValueFromUser() };
17             char  GetOperatorFromUser{ Opperator() };
18             double GetTerm2FromUser{ GetValueFromUser() };
19             if (GetOperatorFromUser == '+' or GetOperatorFromUser == '-' or GetO
    peratorFromUser == '*' or GetOperatorFromUser == '/') {
20                 std::cout << GetTerm1FromUser << " " << GetOperatorFromUser << "
    " << GetTerm2FromUser << " = " << calculator(GetTerm1FromUser,GetOperatorFromUse
21  r,GetTerm2FromUser) << "\n";
22                 std::cout << "\n";
23                 std::cout << "your prgram has ran seccessfully !!!!";
```

```
                std::cout << "\n";
            }
            else {
                std::cout << "\n";
                std::cout << "you entered invalid symbol , something went wrong
 , we couldn't comeplete your operation\n";
                std::cout << "\n";
                std::cout << "compilation failed!!!!\n";
            }
            std::cout << "\n";
        }

        else if (Square_Root == 's') {
            std::cout <<"\n";
            std::cout << "enter your rational number here :\n";
            double radicand;
            std::cin >> radicand;
            std::cout << "the square root of " << radicand << " = " << std::sqrt
(radicand) << "\n";

        }
        else {
            std::cout << "\n";
            std::cout << "please click s or b!!!!! \n";
        }

    }
    return 0;
}
```

MathWork.cpp

```
#include <iostream>

double GetValueFromUser() {

    std::cout << "enter a rational number :\n";
    double input{0.0};
    std::cin >> input;
    return input;
}
char Opperator() {
    std::cout << "enter a valid operator such as'+' and '-' , '/' , '*' : \n";
    char symbol;
    std::cin >> symbol;
    return symbol;
}
double calculator(double term1, char symbol, double term2) {
    if (symbol == '+') {
        std::cout << "\n";
        return term1 + term2;
    }
    else if (symbol == '-') {
        std::cout << "\n";
        return term1 - term2;
    }
    else if (symbol == '*') {
        std::cout << "\n";
        return term1 * term2;
    }
    else if (symbol == '/') {
        std::cout << "\n";
        return term1 / term2;
```

```
32      }
33      return 0;
34  }
```

ForwardDeclaration.h

```
1  #ifndef header // i was actually going to use #pragma once cuz my compiler suppor
2  t this feature
3  //but i'll stick around with this
4  #define header
5  double GetValueFromUser();
6  char Opperator();
7  double calculator(double term1, char symbol, double term2);
   #endif
```

nascardriver
January 31, 2020 at 12:32 am · Reply

That's better already. Explanations to my comments that you didn't implement:

- Use single quotation marks for characters (' ' instead of " ")
"\n" is a string, '\n' is a character. If you use a string, the function you're calling has to do more work and it's taking up more memory.

- Name variables descriptively. x, y, z, q, s, have no meaning.
You did this, but your new names, particularly line 16-18, look like function names because of the "Get" prefix. Without the "Get", they're easier to recognize as variable names.

- Inconsistent formatting. Use your editor's auto-formatting feature.
Please look up how to use the auto-formatter of your editor. It will help you to consistently format your code. That makes your code look better and saves time.

Omran
January 31, 2020 at 12:41 am · Reply

yeah i will work on it even more thank you :)

Omran
January 31, 2020 at 12:47 am · Reply

for auto format feature , i'm using visual studio 2019 , and
i auto format my code by going to edit/advanced/format document
is that right ?

nascardriver
January 31, 2020 at 1:02 am · Reply

I don't have VS, sounds right though.

Omran
January 31, 2020 at 1:13 am · Reply

i'm sorry if i made you tired of me but is this good ?? ,
and please one question am i going to learn how to use macros in this

tutorial :)
and by the way sorry about the auto format feature i didn't know how to do it :)

Main.cpp

```
#include <iostream>
#include "ForwardDeclaration.h"
#include <cmath>

int main() {
    while (true) {

        char Square_Root;
        std::cout << '\n';
        std::cout << "note : click 's' if you want to find the square root of a number ,or click 'b' to begin basic operations : \n";
        std::cin >> Square_Root;
        if (Square_Root== 'b') {
            std::cout << '\n';
            std::cout << "hello , this is a simple calculator\n";
            std::cout << '\n';
            double Term1{ GetValueFromUser() };
            char  Symbol{ Opperator() };
            double Term2{ GetValueFromUser() };
            if (Symbol == '+' or Symbol == '-' or Symbol == '*' or Symbol == '/') {
                std::cout << Term1 << ' ' << Symbol << ' ' << Term2 << '=' << Calculator(Term1,Symbol,Term2) << '\n';
                std::cout << '\n';
                std::cout << "your prgram has ran seccessfully !!!!";
                std::cout << '\n';
            }
            else {
                std::cout << '\n';
                std::cout << "you entered invalid symbol , something went wrong , we couldn't comeplete your operation\n";
                std::cout << '\n';
                std::cout << "compilation failed!!!!\n";
            }
            std::cout << '\n';
        }

        else if (Square_Root == 's') {
            std::cout <<'\n';
            std::cout << "enter your rational number here :\n";
            double radicand;
            std::cin >> radicand;
            std::cout << "the square root of " << radicand << ' ' << '=' << ' ' << std::sqrt(radicand) << '\n';
                std::cout << '\n' << "your program has ran seccessfully!!!!\n";

        }
        else {
            std::cout << '\n';
            std::cout << "please click s or b!!!!! \n";
        }

    }
    return 0;
}
```

MathWork.cpp

```cpp
#include <iostream>

double GetValueFromUser() {

    std::cout << "enter a rational number :\n";
    double input{0.0};
    std::cin >> input;
    return input;
}
char Opperator() {
    std::cout << "enter a valid operator such as'+' and '-' , '/' , '*'
: \n";
    char symbol;
    std::cin >> symbol;
    return symbol;
}
double Calculator(double term1, char symbol, double term2) {
    if (symbol == '+') {
        std::cout << '\n';
        return term1 + term2;
    }
    else if (symbol == '-') {
        std::cout << '\n';
        return term1 - term2;
    }
    else if (symbol == '*') {
        std::cout << '\n';
        return term1 * term2;
    }
    else if (symbol == '/') {
        std::cout << '\n';
        return term1 / term2;
    }
    return 0;
}
```

ForwardDeclaration.h

```cpp
#ifndef header // i was actually going to use #pragma once cuz my compil
er support this feature
//but i'll stick around with this
#define header
double GetValueFromUser();
char Opperator();
double Calculator(double term1, char symbol, double term2);
#endif
```

nascardriver
January 31, 2020 at 1:22 am · Reply

Macros were removed from the lessons because they're not needed anymore. C++ has safer and easier alternatives.

If you want to print multiple characters, use a string. For example here,

```cpp
' ' << '=' << ' '
```

" = " was fine. Only if you want to use single characters, use single quotation marks.

Omran
January 31, 2020 at 1:32 am · Reply

thank you thank you thank you a lot , you helped me improve my code , don't worry i swear i'm not gonna post the code again
, but thank you by the way :D

John
January 16, 2020 at 5:48 am · Reply

Hi, this is my program, could you check and give me feedback, any feedback is welcome, please be brutally honest.
My apologies, I couldn't find out how to put it inside code tags.

// test
[/test]

```
test
```

[test] test [/test]

```
#include <iostream>

int userInput() {

    int number;
    std::cout << "Enter number : " << std::endl;
    std::cin >> number;
    return number;
}

char userOpp() {

    char opperator;
    std::cout << "Enter opperator : " << std::endl;
    std::cin >> opperator;
    return opperator;
}

int calculateResult(int int1, char opperator, int int2) {
    if (opperator == '+') {
        return int1 + int2;
    } else if (opperator == '-') {
        return int1 - int2;
    } else if (opperator == '*') {
        return int1 * int2;
    } else if (opperator == '/') {
        return int1 / int2;
    }
    return 0;
}

int main() {

    char opp;

    while (true) {

        int num1 = userInput();
        char opp = userOpp();
```

```
    int num2 = userInput();

    std::cout << std::endl << num1 << ' ' << opp
    << ' ' << num2 << " = " << calculateResult(num1, opp, num2)
    << std::endl << std::endl;

    char quit;

    std::cout << "Enter q to exit any other character to continue " << std::endl;
    std::cin >> quit;

    if (quit == 'q') {
       break;
    } else {
       ;
    }
    std::cout << std::endl;

  }
  return 0;

}
```

nascardriver
January 16, 2020 at 5:55 am · Reply

[-code]
// your code here
[-/code]
without the -

```
// your code here
```

- Don't use `std::endl` unless you need to. '\n' is faster.
- Initialize your variables with brace initialization, it's safer.
- Avoid abbreviations, they make your code harder to read.
- You don't need an `else` if you don't use it.

Kermit
January 10, 2020 at 11:39 pm · Reply

Hey here is what i did to make it fully working. vid link of testing it out xD
https://youtu.be/hNwfOwA5a4c

```
#include <iostream>

using namespace std; // Alex or nasca ignore this :P

// WE are going to show How to Design a program
// We have to break down our progress a bit by bit
// and do the progress of one thing with function
// For calculator we have to take input from user keyboard twice,
// And an opperator from user

int userInput() // this requires to get input from user keyboard
{
    cout << "Enter a number: ";
    int no{};
    cin >> no;
    return no;
```

```
}

char userOpr()    // Using type char to store the ASCI Opperator means requires char to st
ore +,*,/,-
{
    cout << "Enter opperator: ";
    char opr{};
    cin >> opr;
    return opr;
}

int calculate(int input1, char opper, int input2)
{

    if (opper == '+')
    {
        return input1 + input2;
    }
    else if (opper == '-')
    {
        return input1 - input2;
    }
    else if (opper == '*')
    {
        return input1 * input2;
    }
    else if (opper == '/')
    {
        return input1 / input2;
    }
    return 0;
}

int main()
{
    // get the first no. from user
    // getUserInput();

    // get the math oppperator from the user
    // getMathOpperator();

    // get the second no. from the user
    // getUserInput();

    // calculate the values
    // calculateResults();

    // print out the results
    // printResults();
    cout << "Hello This is a simple calculator program Designed by Kermity!\n\n";
    while (true)
    {

        int input1{ userInput() };
        int input2{ userInput() };
        char opper{ userOpr() };    // Using type char to store the ASCI Opperator means r
equires char to store +,*,/,-


        cout << input1 << " " << opper << " " << input2 << " = " << calculate(input1, oppe
r, input2) << '\n' << '\n';
    }
```

```
77
78
79      return 0;
80
81
    }
```

nascardriver
January 12, 2020 at 3:30 am · Reply

Hey!

Looking good, seems like you already know another language :)
Avoid abbreviations, the time you save writing them is time wasted reading them. Your code is read more often than it's written.

Kermit
January 14, 2020 at 2:34 am · Reply

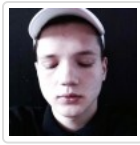
Thanks for the kinds words :)

Omran
January 30, 2020 at 4:14 am · Reply

i just wanted to know what is abbreviation sorry i forgot it T_T

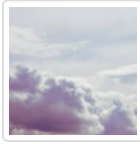
nascardriver
January 30, 2020 at 4:21 am · Reply

A short form of a word, eg. opr instead of operator



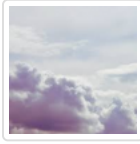
wild boy
December 24, 2019 at 2:26 am · Reply

love u guys



Chayim
December 17, 2019 at 10:32 pm · Reply

When I click on -NEXT- it go’s to page 7.1 and not to 2.x



Chayim
December 17, 2019 at 1:36 am · Reply

Will it make problems for the compiler if the functions are not in order of sequence? Or will it take longer to compile or execute?

nascardriver
December 17, 2019 at 2:26 am · Reply

There's no difference.

Adam
October 3, 2019 at 12:30 pm · Reply

Is that an ok code?

```
#include <iostream>

//  Implementation of the getUserInput function
double getUserInput()
{
    std::cout << "Enter an integer: ";
    double input{ 0 };
    std::cin >> input;

    return input;
}
// Implementation of getMathematicalOperation function
char getMathematicalOperation()
{
    std::cout << "Enter an operator: ";
    char mathoperator{};
    std::cin >> mathoperator;
    return mathoperator;
}
// Implementation of calculateResult() function
double calculateResult(double x, char y, double z)
{
    if (y == '-')
    {
        return x - z;
    }
    if (y == '+')
    {
        return x + z;
    }
    if (y == '*')
    {
        return x * z;
    }
    if (y == '/')
    {
        return x / z;
    }
}
    //Implementation of printResult function
void printResult()
{
    double value1{ getUserInput() };
    char useroperator{ getMathematicalOperation() };
    double value2{ getUserInput() };
    double result{ calculateResult(value1, useroperator, value2) };
    std::cout << "The result of " << value1 << useroperator << value2 << " is: " << result
<< '\n';
}

int main()
{
    for (;;)
    {
        printResult();
```

```
    }
    return 0;
}
```

nascardriver
October 4, 2019 at 1:47 am · Reply

- Use double literals for doubles (0.0 instead of 0).
- Don't use comments to describe what you do, use them to describe why you do it.
- Line 27, 31, 35: Should be `else if`. I don't know if it was covered yet.
- Line 52: Should be `while (true)`.
- "printResult" is a poor name. The function does more than printing a result.
- `calculateResult` is missing a `return` in case `y` doesn't match any of the operators.
- Name your variables descriptively. "y" doesn't provide any information about the variable. You can re-use "useroperator".

Looks good otherwise, keep on sharing your solutions :)

Shubham Maske
September 27, 2019 at 9:38 am · Reply

guys why I am getting the error can you help me out I have been trying to figure out but no result so far.

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

int firstNum();
char myOperator();
double  totalSum();

int main()
{
    int x  = firstNum();
    char z = myOperator();
    int y  = firstNum();

    cout<< x << z <<  y << '=' << totalSum() ;
    return 0;
}

int firstNum( )
{
    cout<< 'enter an integer ' ;
    int a{0};
    cin>> a;
    return a;
}

char myOperator()
{
    cout<< 'enter an operator' ;
    char b{0};
    cin>> b;
    return b;

}

```

```
double totalSum(int a , char b , int z)
{


    if(b == '+')
        return a + z;

    if(b == '-')
        return a - z;

    if(b == '/')
        return a / z;

    if (b == '*')
        return a * z;
}
```

nascardriver
September 28, 2019 at 12:36 am · Reply

When you get an error, post the error message.
Re-read lesson 2.3. You're not passing arguments to `totalSum`.

- Initialize your variables with brace initializers.
- `totalSum` is missing a `return`.
- Don't use `using namespace`.
- Use your editor's auto-formatting feature.
- Name your variables descriptively.
- If your program prints anything, the last thing it prints should be a line-feed.
- `totalSum` implicitly casts an `int` to `double`. You'll never get floating point results. You need to cast the operands before you divide.

Slyde
July 10, 2019 at 12:32 am · Reply

Hello Alex. I guess this will be covered at some point. But since I ran in to it this evening, I thought I'd go ahead and ask abt it. I'm able to catch invalid entries in my getOperator(); function through an IF statement and recursion (I hope I'm using that term in the right context here). After a few bad entries, I'll give it one it accepts. But no matter what correct entry I give, it keeps pulling out the * sign to use. This only happens when I put in garbage before entering a correct operator.

```
#include <iostream>


int getNum();

char getOperator();

void totalNums(int x, int y, char z);


int main()
{
    int firstNum{ getNum() };
    char oper{ getOperator() };
    int secondNum{ getNum() };

    totalNums(firstNum, secondNum, oper);
```

```
18
19      return 0;
20  }
21
22  int getNum()
23  {
24      std::cout << "Enter a number: ";
25      int x{};
26      std::cin >> x;
27
28      return x;
29  }
30
31  char getOperator()
32  {
33      std::cout << "Choose an Operator (+ - / *): ";
34      char a = ' ';
35      std::cin >> a;
36
37      if (a != '+' && a != '-' && a != '/' && a != '*')
38          getOperator();
39
40      return a;
41  }
42
43  void totalNums(int x, int y, char z)
44  {
45      if (z == '+')
46          std::cout << x << " + " << y << " = " << x + y << "\n\n";
47      else if (z == '-')
48          std::cout << x << " - " << y << " = " << x - y << "\n\n";
49      else if (z == '/')
50          std::cout << x << " / " << y << " = " << x / y << "\n\n";
51      else
52          std::cout << x << " * " << y << " = " << x * y << "\n\n";
53  }
```

Thanks for your help.

abdulilah
July 10, 2019 at 3:05 am · Reply

This fixed it for me , I think its something with the flow but I am new to c++.
I think its going into getOperator() every time until u get the right operator, but the is still coming from the first time the function was called.
The * you have been getting is probably the default char. Like 0 is the default int if u enter something else into an int value.
Hope this helps.

```
1   char getOperator()
2   {
3       std::cout << "Choose an Operator (+ - / *): ";
4       char a = ' ';
5       std::cin >> a;
6
7       if (a != '+' && a != '-' && a != '/' && a != '*')
8           a = getOperator();
9
10      return a;
11  }
```

nascardriver
July 10, 2019 at 5:02 am · Reply

- Initialize your variables with brace initializers.
- Only initialize variables to a specific value if you need that value. Use empty curly braces otherwise.
- Name your variables descriptively.

@abdulilah is partially right. You're not doing anything with the value returned by `getOperator` in line 38. Either assign to `a` or return immediately.
Without it, `a` is still ' ' in line 40 and * is your default operator in line 51-52.

Slyde
July 10, 2019 at 12:51 pm · Reply

Thank you both. I was thinking it was being reset to a NULL value each iteration on line 34: char a = ' ';. That's what was throwing me off. Thanks again for the help.

```
1  char getOperator()
2  {
3      std::cout << "Choose an Operator (+ - / *): ";
4      char getOper{};
5      std::cin >> getOper;
6
7      if (getOper != '+' && getOper != '-' && getOper != '/' && getOper != '*')
8          getOper = getOperator();
9
10     return getOper;
11 }
```

Works like a charm now.

nascardriver
July 11, 2019 at 5:16 am · Reply

```
1 char ch{ ' ' }; // space
2 char ch{ '\x00' }; // 0
3 char ch{ '\0' }; // 0
4 char ch{ 0 }; // 0
5 char ch{}; // 0
```

Mike
September 13, 2019 at 1:35 pm · Reply

I used your code verbatim, even with your changes, and I still get the * after typing a non entry selection followed by a correct entry.

I added two lines to the code to help me determine where/when the value is being changed back to the original invalid entry.

In the getOperator(), I added it right before the return.
In the main(), I added it right before the totalNums().

The value remains correct up to the return in getOperator. The + operator is displayed correctly

Yet when it returns to the main(), the value is now recorded as the original invalid entry.

How is this so? How can it return an invalid entry when the If statement doesn't even allow the code to reach the return command until a correct entry is received? So I don't understand how the invalid entry even gets transferred out of the function. Obviously, it is though, and it's probably something simple. So I can't wait to have my mine blown.

```
1  int main()
2  {
3      int firstNum{ getNum() };
4      char oper{ getOperator() };
5      int secondNum{ getNum() };
6
7      std::cout << oper << '\n';  // added to determine current value. Value c
   hanges back here for some reason. Why?
8
9      totalNums(firstNum, secondNum, oper);
10
11 }
12
13 char getOperator()
14 {
15     std::cout << "Choose an Operator (+ - / *): ";
16     char getOper{};
17     std::cin >> getOper;
18
19     if (getOper != '+' && getOper != '-' && getOper != '/' && getOper != '*'
   )
21         getOperator();
22
23     std::cout << getOper << '\n'; // added to determine current value. Value
   is correctly identified as a + operator
24
25     return getOper;
   }
```

nascardriver
September 14, 2019 at 12:59 am · Reply

You're not using the return value of `getOperator` in line 20. `getOper` of the first call stays the same. The results of all recursive calls are discarded.

Mike
September 14, 2019 at 9:27 am · Reply

Thanks again for replying! I really don't know how you have the time or even the patience for beginners like me.

Ok, so you state the value for 'getOper' stays the same for the first call. But I'm confused (imagine that). So if there is no return value for 'getOperator' in line 20, then how does the caller even receive the value from 'getOper' for the invalid entry during its first call?

This is a general outline of how I see the code's logic during execution? Please show me where I'm misinterpreting it, and remember I'm a beginner, so any clarification is always welcomed.

1. Line 28: Choose an Operator: First time, I will enter invalid entry, like $
2. Line 30: The getOper variable now stores this invalid entry with a $

3. If statement determines the getOper value is invalid, and thus calls the getOperator()
4. Back to Line 28: Choose an Operator: this time I choose a valid entry, like +
5. Line 30: This time getOper stores the correct value, +
6. Line 35: (for debugging) Confirms and displays the correct value for getOper, +
7. Line 37: Returns the + value of getOper to the caller, or should anyways
8. Line 10: Return value is stored in 'oper', BUT for some reason it's back to $
9. Line 13: (for debugging): Confirms the invalid entry by displaying the $

nascardriver
September 15, 2019 at 4:15 am · Reply

8. is where you're wrong. This is the return of the second call. When the second call returns, execution continues after line 20 of the first call.
The returned value of the second call isn't used.
Every time you call `getOperator`, a new `getOper` is created. I'll append numbers to the variables and calls so you can see the difference.

1. Call to `getOperator1` from `main`.
2. Invalid entry ($), stored in `getOper1`.
3. Call to `getOperator2` from `getOperator1`
4. Valid entry (+), stored in `getOper2`.
5. `getOper2` is returned.
6. Execution continues in `getOperator1`, after line 20 (Note: The returned value isn't used, `getOper1` is still '$').
7. `getOperator1` returns `getOper1` ($)

You can mark functions as `nodiscard`

```
1 [[nodiscard]] char getOperator()
2 {
3   // ...
4 }
```

Now when you call `getOperator` and don't use the return value, you'll get a warning (Your compiler is allowed to ignore the attribute, but most compilers will issue a warning).

Mike
September 15, 2019 at 8:01 am · Reply

Thank you for the detailed outline and appending the numbers. It always helps to have some type of visual explanation. This really helped me to see how it is actually being processed/parsed or whatever you call it.

I had no idea the getOperator was being incremented or being considered as a separate instance of itself as you described for each time it is being called. I assume this is only because it is calling itself from within itself. I don't believe this has been covered yet in the tutorials, though if so, could you kindly point me to it so I can read more about it?

Thanks again!

nascardriver
September 15, 2019 at 8:03 am · Reply

It's covered later. It's called _recursion_, and it's bad.

Mike
September 15, 2019 at 8:14 am

So, I assume it would be better to use something like Do/While in this function, rather than recalling the same function within itself?

nascardriver
September 15, 2019 at 8:15 am · Reply

Exactly, loops are better.

aymnomous
July 4, 2019 at 8:43 am · Reply

Can someone explain in section "Implementation step 2: Implement each function" what the Define the function prototype (inputs and outputs) means? Specifically the input and output part.

nascardriver
July 4, 2019 at 8:48 am · Reply

Return type and parameters

alfonso
May 11, 2019 at 1:26 am · Reply

"Clean the house
Vacuum the carpets
Clean the bathrooms
Scrub the toilet (yuck!)
Wash the sink
Clean the kitchen
Clear the countertops
Clean the countertops
Scrub the sink
Take out the trash

Now we have a hierarchy of tasks, none of them particularly hard."

I read it : Now we have a hierarchy of tasks, ONE of them is particularly hard.
I thought - hmm, that one must be the "yuck!" one :)

Li
April 22, 2019 at 2:37 pm · Reply

I pretty much agree with the design philosophy expressed here. The one point I mostly disagree with is the "Focus on one area at a time." paragraph. I agree that you need to focus on one part at

a time, once it's working at all. I very much don't agree that you should make that part "fully working" or take that part "through to completion". When you sit down to design your program, it is possible that you will spend enough time and effort to completely (and correctly) define the required program structure, data types, and needed processing, I/O and storage. Unlikely, but possible. The "right way" (arguably) to look at all but the simplest programs is that the process of creating them is a development process. It is iterative, with a lot of feedback (i.e. trial and error). There will almost certainly be some evolution of your understanding about program structure, data types, processing and I/O as you fill out the details. Meaning there will almost certainly be things you need to change/correct/improve. So, rather than "fully" finish any one part of your program before moving on, I'd say MINIMALLY finish it. Get enough of it together and working and then move on to the next area. Build up its complexity and 'polish' while testing it. You want to spend the least time possible to get to the next "level" and that "level" might be a poor imitation of what the end product will be. Another way to describe this is to work on the 'pieces' one at a time, and improve them sequentially and incrementally. The question is how big of a increment, how much effort, to spend on one piece before moving to the next. I'd say use the law of diminishing returns. Do the obvious stuff to improve a component, test, and move on. To repeat, do NOT spend the time to fully perfect each individual part. Perfect is the enemy of the good.

Alex
April 23, 2019 at 5:01 pm · Reply

Agreed. Lesson amended. Thanks for the feedback!

James
March 31, 2019 at 1:26 pm · Reply

I have been learning the language of C++ for the past two months. I have managed to code a rudimentary calculator.

Firstly, I tried to code the calculator so that the user is forced to enter 5 numbers.

The code for this works without any errors.

After this, I tried to give the user the option to stop at only two numbers, but for some reason, this part of the code is not working.

When I compile the code for the calculator, it works without any errors but the code I have written for the makeDecision(), makeDecision2(), and makeDecision3() functions (the part of the code that give the user the option to only enter two numbers) are getting ignored.

Can you please tell me how to fix this problem?

Here is the code:

```
#include <iostream>
#include <stdio.h>
#include <cstdio>

int getNumbers()
{
    std::cout << "Please type any integer: ";
    int x{ 0 };
    std::cin >> x;
    return x;
}

int getOperators()
{
    int mop{ 0 };
    do
    {
```

```
        std::cout << "Please type the operator you'd like for your calculation (+, -, *, /): ";
        std::cin >> mop;

        if (mop <= 0 || mop > 4)
        {
            std::cout << "The number you entered isn't one of the options provided. Please input a proper number for the calculator.\n";
        }

    } while (mop <= 0 || mop > 4);

    return mop;
}

int getOperatorChar()
{
    char ops = ' ';
    bool cont = true;
    int mop = 0;

    do
    {
        printf("Please type the operator you'd like for your calculation (+, -, *, /): ");
        std::cin >> ops;

        if (ops == '+')
            mop = 1;

        else if (ops == '-')
            mop = 2;

        else if (ops == '*')
            mop = 3;

        else if (ops == '/')
            mop = 4;

        else
        {
            std::cout << "The number you entered isn't one of the options provided. Please input a proper number for the calculator.\n";
        }

    } while (mop <= 0 || mop > 4);

    return mop;
}


int makeDecision()
{
    int yesno{ 0 };
    std::cout << "Would you like to put another integer in your equation? Please answer with Yes or No. \n";
    std::cin >> yesno;
    return yesno;
}

int makeDecision2()
{
```

```
    char dec = ' ';
    bool cont = true;
    int yesno = 0;

    do
    {
        printf("Would you like to put another integer in your equation? Please answer wit
h Yes or No. ");
        std::cin >> dec;

        if (dec == 'Yes')
            yesno = 1;

        else if (dec == 'No')
            yesno = 2;
    }

    while (dec <= 0 || dec > 2);

    {
        std::cout << "The answer you entered isn't part of the options provided. Please i
nput a proper answer to continue. \n";
    }

    return yesno;
}

int makeDecision3()
{
    int yesno{ 0 };
    int answer{ 0 };

    if (yesno == 1)
    return getOperators();

    if (yesno == 2)
        return answer;

    return 0;
}

int getAnswer(int x, int mop, int y)
{
    if (mop == 1)
        return x + y;

    if (mop == 2)
        return x - y;

    if (mop == 3)
        return x * y;

    if (mop == 4)
        return x / y;

    return 0;
}

void printResult(int answer)
{
    std::cout << "The answer to your chosen equation is " << answer << "\n";
}
```

```
139
140 int main()
141 {
142     std::cout << "Hello! This program is a calculator that allows you to add, subtract, m
143 ultiply, and/or divide five integers!" << '\n' << '\n';
144
145     int num1 = getNumbers();
146
147     int mop1 = getOperatorChar();
148
149     int num2 = getNumbers();
150
151     int makeDecision();
152
153     int makeDecision2();
154
155     int makeDecision3();
156
157     int mop2 = getOperatorChar();
158
159     int num3 = getNumbers();
160
161     int mop3 = getOperatorChar();
162
163     int num4 = getNumbers();
164
165     int mop4 = getOperatorChar();
166
167     int num5 = getNumbers();
168
169     int answer = getAnswer(num1, mop1, num2);
170
171     int answer2 = getAnswer(answer, mop2, num3);
172
173     int answer3 = getAnswer(answer2, mop3, num4);
174
175     int answer4 = getAnswer(answer3, mop4, num5);

    printResult(answer4);

    return 0;
}
```

nascardriver
April 1, 2019 at 2:42 am · Reply

* Line 33, 34, 35, 75, 76, 77, 140, 142, 144, 152, 154, 156, 158, 160, 162, 164, 166, 168, 170: Initialize your variables with brace initializers. You used copy initialization.
* Line 39: Initialize your variables with brace initializers. You used direct initialization.
* Line 18, 23, 39, 56, 68, 81, 94, 138: Limit your lines to 80 characters in length for better readability on small displays.
* Enable compiler warnings and fix them. Lesson 0.10, 0.11.

You can't store strings in ints or chars. See lesson 4.4b.
Line 146-150 aren't calling the functions. Remove the "int" in front of them. Lesson 2.7.

Louis Cloete
April 26, 2019 at 12:48 pm · Reply

To add to @nascardriver's comments:

1. You seem to know about the bool type. Don't use integers for YesNo. Use bools and let Yes be true and No be false. (see also chapter 4 for more on bool.)
2. Don't use printf() to print output. Use std::cout. There is no good reason to mix and match printf() and std::cout in C++. You can always use std::cout. It is also simpler and less error prone.

Liquid
March 27, 2019 at 6:47 am · Reply

I have a little problem with my program, it executes as it should but the outcome is not right.

```
#include <iostream>


int getNumberFromUser()
{
    std::cout << "Enter Integer: ";
    int input;
    std::cin >> input;

    return input;
}

char getOperatorFromUser()
{
    std::cout << "Enter Operator: ";
    char input;
    std::cin >> input;

    return input;
}

int getEquasion(int x,char y, int z)
{
    //testing
     char str1 = '+';
     char str2 = '-';
     char str3 = '*';
     char str4 = '/';

    if (y == str1)
        return x + y;

    if (y == str2)
        return x - y;

    if (y == str3)
        return x * y;

    if (y == str4)
        return x / y;

    return 0;
}

int main()
{
        int x = getNumberFromUser(); //gets number from user and stores it in x.
        char y = getOperatorFromUser(); //gets Operator from user and stores it in y.
        int z = getNumberFromUser(); //gets number from user and stores it in z.
```

```
50
51      std::cout << x << y << z << "=" << getEquasion(x, y, z) << '\n'; //prints equasion.
52
53      return 0;
54  }
```

nascardriver
March 27, 2019 at 9:03 am · Reply

* Line 25, 26, 27, 28, 47, 48, 49: Initialize your variables with brace initializers. You used copy initialization.
* Line 7, 16: Initialize your variables with brace initializers.
* Line 48, 51: Limit your lines to 80 characters in length for better readability on small displays.

Enable compiler warnings (Lesson 0.10, 0.11). @z is unused in @getEquasion.

Liquid
March 27, 2019 at 9:57 am · Reply

Wow, Thank you ^^ forgot to turn the warnings back on after reinstalling.
what a stupid mistake now that I think about it ^^

Well thank you for your great answer ^^

Rae
March 26, 2019 at 2:53 am · Reply

This is beautifully explained. :)

Juan
March 9, 2019 at 5:30 pm · Reply

Is excellent know where to start!

« Older Comments 1 ... 3 4 5