1.6 — Uninitialized variables and undefined behavior

BY ALEX ON FEBRUARY 1ST, 2019 | LAST MODIFIED BY NASCARDRIVER ON NOVEMBER 13TH, 2019

Uninitialized variables

Unlike some programming languages, C/C++ does not initialize most variables to a given value (such as zero) automatically. Thus when a variable is assigned a memory location by the compiler, the default value of that variable is whatever (garbage) value happens to already be in that memory location! A variable that has not been given a known value (usually through initialization or assignment) is called an **uninitialized variable**.

Author's note

Many readers expect the terms “initialized” and “uninitialized” to be strict opposites, but they aren’t quite! Initialization means the object was provided with an initial value at the point of definition. Uninitialized means the object has not been given a known value (through any means, including assignment). Therefore, an object that is not initialized but is then assigned a value is no longer *uninitialized* (because it has been given a known value).

To recap:

- Initialization = The object is given a known value at the point of definition.
- Assignment = The object is given a known value beyond the point of definition.
- Uninitialized = The object has not been given a known value yet.

As an aside...

This lack of initialization is a performance optimization inherited from C, back when computers were slow. Imagine a case where you were going to read in 100,000 values from a file. In such case, you might create 100,000 variables, then fill them with data from the file.

If C++ initialized all of those variables with default values upon creation, this would result in 100,000 initializations (which would be slow), and for little benefit (since you’re overwriting those values anyway).

For now, you should always initialize your variables because the cost of doing so is miniscule compared to the benefit. Once you are more comfortable with the language, there may be certain cases where you omit the initialization for optimization purposes. But this should always be done selectively and intentionally.

Using the values of uninitialized variables can lead to unexpected results. Consider the following short program:

```
#include <iostream>

int main()
{
    // define an integer variable named x
    int x; // this variable is uninitialized because we haven't given it a value

    // print the value of x to the screen
    std::cout << x; // who knows what we'll get, because x is uninitialized

    return 0;
}
```

In this case, the computer will assign some unused memory to *x*. It will then send the value residing in that memory location to *std::cout*, which will print the value (interpreted as an integer). But what value will it print? The answer is "who knows!", and the answer may (or may not) change every time you run the program. When the author ran this program in Visual Studio, *std::cout* printed the value 7177728 one time, and 5277592 the next. Feel free to compile and run the program yourself (your computer won't explode).

As an aside...

Some compilers, such as Visual Studio, *will* initialize the contents of memory to some preset value when you're using a debug build configuration. This will not happen when using a release build configuration. Therefore, if you want to run the above program yourself, make sure you're using a *release build configuration* (see lesson **0.9 -- Configuring your compiler: Build configurations** for a reminder on how to do that). For example, if you run the above program in a Visual Studio debug configuration, it will consistently print -858993460, because that's the value (interpreted as an integer) that Visual Studio initializes memory with in debug configurations.

Most modern compilers will attempt to detect if a variable is being used without being given a value. If they are able to detect this, they will generally issue a compile-time error. For example, compiling the above program on Visual Studio produced the following warning:

```
c:\VCprojects\test\test.cpp(11) : warning C4700: uninitialized local variable 'x' used
```

If your compiler won't let you compile and run the above program for this reason, here is a possible solution to get around this issue:

```
#include <iostream>

void doNothing(const int &x) // Don't worry about what &x is for now, we're just using it to t
{
}

int main()
{
    // define an integer variable named x
    int x; // this variable is uninitialized

    doNothing(x); // make compiler think we're using this variable

    // print the value of x to the screen (who knows what we'll get, because x is uninitialize
    std::cout << x;

    return 0;
}
```

Using uninitialized variables is one of the most common mistakes that novice programmers make, and unfortunately, it can also be one of the most challenging to debug (because the program may run fine anyway if the uninitialized value happened to get assigned to a spot of memory that had a reasonable value in it, like 0).

This is the primary reason for the "always initialize your variables" best practice.

Undefined behavior

Using the value from an uninitialized variable is our first example of undefined behavior. **Undefined behavior** is the result of executing code whose behavior is not well defined by the C++ language. In this case, the C++ language doesn't have any rules determining what happens if you use the value of a variable that has not been given a known value. Consequently, if you actually do this, undefined behavior will result.

Code implementing undefined behavior may exhibit *any* of the following symptoms:

- Your program produces different results every time it is run.
- Your program consistently produces the same incorrect result.
- Your program behaves inconsistently (sometimes produces the correct result, sometimes not).
- Your program seems like its working but produces incorrect results later in the program.
- Your program crashes, either immediately or later.
- Your program works on some compilers but not others.
- Your program works until you change some other seemingly unrelated code.

Or, your code may actually produce the correct behavior anyway. The nature of undefined behavior is that you never quite know what you're going to get, whether you'll get it every time, and whether that behavior will change when you make other changes.

C++ contains many cases that can result in undefined behavior if you're not careful. We'll point these out in future lessons whenever we encounter them. Take note of where these cases are and make sure you avoid them.

Rule

Take care to avoid all situations that result in undefined behavior, such as using uninitialized variables.

Author's note

One of the most common types of comment we get from readers says, "You said I couldn't do X, but I did it anyway and my program works! Why?".

There are two common answers. The most common answer is that your program is actually exhibiting undefined behavior, but that undefined behavior just happens to be producing the result you wanted anyway... for now. Tomorrow (or on another compiler or machine) it might not.

Alternatively, sometimes compiler authors take liberties with the language requirements when those requirements may be more restrictive than needed. For example, the standard may say, "you must do X before Y", but a compiler author may feel that's unnecessary, and make Y work even if you don't do X first. This shouldn't affect the operation of correctly written programs, but may cause incorrectly written programs to work anyway. So an alternate answer to the above question is that your compiler may simply be not following the standard! It happens. You can avoid much of this by making sure you've turned compiler extensions off, as described in lesson **0.10 -- Configuring your compiler: Compiler extensions**.

Quiz time

Question #1

What is an uninitialized variable? Why should you avoid using them?

Show Solution

Question #2

What is undefined behavior, and what can happen if you do something that exhibits undefined behavior?

Show Solution


1.7 -- Keywords and naming identifiers


Index


1.5 -- Introduction to iostream: cout, cin, and endl

C++ TUTORIAL | PRINT THIS POST

48 comments to 1.6 — Uninitialized variables and undefined behavior

Vitaliy Sh.
January 12, 2020 at 9:04 pm · Reply

Hi Sires!

"Therefore, an object that is not initialized but is then assigned a value is no longer uninitialized (because it has been given a known value)."
** smth: ``value). But the object not becomes initialized because of this.``
** https://www.learncpp.com/cpp-tutorial/uninitialized-variables-and-undefined-behavior/#comment-406312

"If your compiler won't let you compile and run the above program for this reason, here is a possible solution to get around this issue:"
****
Is there no other way for me, but disable the -Werror?

```
# [[ ${1} =~ .*\.cpp ]] || { ... ; exit 1; } at the top.

### ./g++Debug.sh"
debug="
     -Og
     -g"

```

```
### ./g++Release.sh
debug="
    -O2
    -flto"

### Both
warnings="
    -Wall
    -Wextra
    -Weffc++
    -Wconversion
    -Wsign-conversion
    -Wpedantic
    -pedantic-errors
    -Werror"

flags="${debug} ${warnings} -std=c++17"

g++ {flags} ${@} -o executables/${1%\.cpp}
```

"Debug":

```
5-actual_UB.cpp:3:27: error: unused parameter ‘x’ [-Werror=unused-parameter]

5-actual_UB.cpp:15:15: error: ‘x’ is used uninitialized in this function [-Werror=uninitiali
```

"Release":

```
5-actual_UB.cpp:3:27: error: unused parameter ‘x’ [-Werror=unused-parameter]
```

if move std::cout to doNothing(const int &x):
"Debug": output is always zero (so clever, g++...)
"Release": linker, lto-wrapper failed (with fno-lto the same):

```
5-actual_UB.cpp:6:15: error: ‘x’ is used uninitialized in this function [-Werror=uninitializ
```

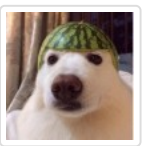
Thanks for all the fun!
****

"Your program seems like its working but produces incorrect results later in the program."
** ``later.`` (minus- "in the program")?

nascardriver
January 13, 2020 at 2:09 am · Reply

```
// no name, no warning, no error
void fn(int){}

// tell the compiler that we don't want to use i
// probably no warning, no error
void fn([[maybe_unused]] int i){}
```



Petertheminer11
January 1, 2020 at 7:50 am · Reply

I'm still a little confused about where the computer is getting its undefined values from, how can it just have random stored values from std::cout, why does the compiler even bother to give you a return value other than 0 when you run the program?

nascardriver
January 1, 2020 at 7:54 am · Reply

When you create a variable, it's stored at some memory location. If you don't initialize the variable, the variable will have whatever value was previously stored at that location (Memory gets reused).

Peterthеminer11
December 31, 2019 at 9:56 am · Reply

#include <iostream>

```
int main()
{
    // define an integer variable named x
    int x; // this variable is uninitialized because we haven't given it a value

    // print the value of x to the screen
    std::cout << x; // who knows what we'll get, because x is uninitialized

    return 0;
}
```

When I ran this it just returned 0

nascardriver
January 1, 2020 at 1:32 am · Reply

Undefined behavior means anything could happen, including seemingly expected behavior.

XAKEP
January 24, 2020 at 2:18 am · Reply

You could be running in debug mode. You'll then get an automatically assigned value.

Marvel S
December 5, 2019 at 11:20 am · Reply

what is meaning of this statement- "Your program works until you change some other seemingly unrelated code." ?

nascardriver
December 6, 2019 at 2:56 am · Reply

Your code works.
You change something.
Something that's unrelated to what you changed breaks.

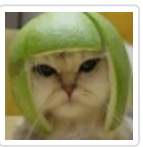
chai
November 26, 2019 at 5:01 am · Reply

doNothing() demonstrating uninitialised variable effects is not working for me. I get c2220 warning treated as error - no 'object' file generated. c4100 'x' unreferenced formal parameter.

Maybe the compiler has been updated to recognise the risk?

I then put std::cout inside doNothing() and get a printout of x being 0.

```
1  include <iostream>
2
3  void doNothing(const int& x) // Don't worry about what &x is for now, we're just using it t
4  {
5
6      std::cout << "x in doNothing is " << x;
7  }
8
9  int main()
10 {
11     // define an integer variable named x
12     int x{}; // this variable is uninitialized
13
14     doNothing(x); // make compiler think we're using this variable
15
16     // print the value of x to the screen (who knows what we'll get, because x is uninitial
17     std::cout << " x in main is " << x << std::endl;
18
19     return 0;
20 }
```



Alex
November 26, 2019 at 1:01 pm · Reply

Try changing:
`void doNothing(const int& x)`
to:
`void doNothing(const int&)`

Does that remove the warning?

If you're running in debug mode, x will be initialized to 0. This initialization to 0 won't happen in non-debug mode, but it's possible it could have value 0 anyway.

Ikenna
November 26, 2019 at 4:02 am · Reply

This is the best C++ tutorial set I have ever seen!!

Seriously, I've taken lots of courses on c++ but none ever explain concepts in great detail such as the difference between initialization, assignment and uninitialized.

Nice work Alex & others. :)

fucitol
November 12, 2019 at 1:15 pm · Reply

in the authors note

Initialization = The object is given an known value at the point of definition.

why is there "an known value" else of "a known value"

IDK

...BTW the website is amazing, i'm learning a lot .....

nascardriver
November 13, 2019 at 3:42 am · Reply

Fixed, thanks!

trisha
November 7, 2019 at 10:36 pm · Reply

void
doNothing
const
int &x

what does this mean?

nascardriver
November 9, 2019 at 2:42 am · Reply

You don't need to know now. Alex just used it so his compiler doesn't error out. You'll learn what it is later.

Ronnie
September 3, 2019 at 7:21 pm · Reply

Hello,
I'm new to coding using Code::Blocks and cannot get the program to compile, even when using the workaround solution and changing to release mode.

Is there something I'm missing?

Thanks for your help.

moly7x
November 12, 2019 at 9:58 am · Reply

U have to download mingw version, "codeblocks-17.12mingw"

Yiğit
August 25, 2019 at 6:22 pm · Reply

Um... I have question like when I do your questions %99 I can't do it like I mean ı can but you are saying more techicall things.. So sometimes I don't know how to explain but after see the answer I am saying to myself "oo, yes ıf you do like that blahbla" ı mean after I saw the answers I can understand is it normall I am beginner btw for programming

nascardriver
August 26, 2019 at 7:57 am · Reply

Yes, that's normal.

Yiğit
August 26, 2019 at 1:15 pm · Reply

Idk man because for applications you need to write a lot of code and hard IDK why should I learn c++ ????????

Ana
September 18, 2019 at 5:54 am · Reply

No one is forcing you, you don't have to learn it if you don't want to. If you stick with it, even just doing 10 minutes a day, you'll learn. You'd be surprised how much you can learn just by chipping away at something.
Remember, start small. Make a simple calculator, just by asking the user to input two numbers and then asking them if they want to add, subtract, etc. No need to worry about GUI or making fancy applications. Everyone has day one, where they suck at the thing they're trying to do. Mozart had day one. Eric Clapton had day one. All the greats in music and film had day one. If you stick with it, and you want to learn, you'll learn. Just don't overwhelm yourself and take things one step at a time. :)

Vulryn
August 15, 2019 at 5:35 am · Reply

I am using do nothing version of this code and I kept getting -858993460. So i changed it from debug to release. Now i keep getting 0 instead of random numbers as output. Is this a problem or undefined behavior ? How can i fix it ?

nascardriver
August 15, 2019 at 6:26 am · Reply

It's undefined behavior. Don't use uninitialized variables.

Juglugs
October 18, 2019 at 3:57 am · Reply

Really, that's your compiler helping you out by initialising the variable to 0 in the background for you - but you can't trust it to do that every time or across different machines, so always initialise your variables

nascardriver
June 1, 2019 at 11:39 pm · Reply

Hi Alex!

Missing "the"
"determining what happens if you use [the] value of a variable"

Alex
June 11, 2019 at 2:08 pm · Reply

Fixed! Thanks for pointing out the omission.

Rhylee
June 1, 2019 at 9:40 pm · Reply

"lack of initialization is a performance optimization that was inherited from C, back when computers were slow. Imagine a case where you were going to read in 100,000 values from a file. In such case, you might create 100,000 variables, then fill them with data from the file. If C++ initialized all of those variables with default values upon creation, this would result in 100,000 initialization (which would be slow), and for little benefit (since you're overwriting those values anyway)"

Isn't that what we were supposed to do anyway? Initialize all variables? yet you wrote it would be slow?
Does this mean the less initialization - the better the computer/program will perform.

nascardriver
June 1, 2019 at 11:44 pm · Reply

For now, initializing all variables is better. Once you've grown confident with the language, you can omit certain initializations.

Juglugs
October 18, 2019 at 4:01 am · Reply

Computers have advanced so much since C was written that the performance 'hit' of initialising variables is so negligible that you won't notice it at our novice level of programming - and probably not even when you're a pro coder! :)
It's best practice to initialise your variables...

Oh, and comment your code!!

Thorick Chow
December 30, 2019 at 11:36 am · Reply

what about an embedded device with a limited CPU and resources ?
C/C++ is a language used in things other than servers, desktops and laptops.

Mabel
June 1, 2019 at 7:42 am · Reply

As we have already been through the chapters of Initializing (1.4), is this chapter of "Uninitialized" basically letting us know how important it is for us to initialize each variable?

Alex
June 1, 2019 at 10:57 am · Reply

The main point is actually to talk about undefined behavior -- which can occur in many ways in C++. Trying to read from an undefined variable is the first case of undefined behavior that new programmers typically encounter, so it makes for a convenient example.

Mabel
June 1, 2019 at 9:27 pm · Reply

Ahhhhhh I get it, Thank you

Amanda
June 1, 2019 at 6:43 am · Reply

"Lack of initialization is a performance optimization. Imagine you needed 10,000 variables. If these variables were initialized by default, all 10,000 would need to be initialized to the default value upon creation." What does that mean exactly? Does it mean the 10,000 variables each have to be initialized?... Because the paragraph below says

"This might result in slow program start up. If you did not need these variables immediately, you could always initialize them as you need them, spreading the initialization cost out over the running time of your program." Im not too sure what it means all together as it says the program may have a slow start up.

Alex
June 1, 2019 at 10:55 am · Reply

I rewrote the section with an easier to follow example. See if it makes more sense now. Thanks for the feedback.

Chayim Eliazer
May 25, 2019 at 7:36 am · Reply

I ran this "uninitialized variable" in Visual Studio 2019 Enterprise and it did not compile it because of error.
"Severity Code Description Project File Line Suppression State
Error MSB6006 "CL.exe" exited with code 2. ConsoleApplication2 C:\Program Files (x86)\Microsoft Visual Studio\2019\Community\MSBuild\Microsoft\VC\v160\Microsoft.CppCommon.targets 429"

Then tried another time and got 2 errors
Severity Code Description Project File Line Suppression State
Error C4700 uninitialized local variable 'x'
used ConsoleApplication2 C:\Users\Chayi\source\repos\ConsoleApplication2\ConsoleApplication2.cpp 11

Severity Code Description Project File Line Suppression State
Error LNK1257 code generation
failed ConsoleApplication2 C:\Users\Chayi\source\repos\ConsoleApplication2\LINK 1

Alex
May 29, 2019 at 12:48 pm · Reply

See the second example in the lesson, which shows a way to work around this.

alfonso
April 24, 2019 at 3:45 am · Reply

1.6 — Uninitialized variables and undefined behavior

Question #1

What is an uninitialized variable?

Hide Solution
An uninitialized variable is a variable that has not been given a value by the program (generally through initialization OR ASSIGNMENT).

BUT

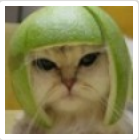
1.4 — Variable assignment and initialization

Question #1

What is the difference between initialization and assignment?

Hide Solution
Initialization gives a variable a value AT THE POINT when it is created. Assignment gives a variable a value AT SOME POINT AFTER it is created.

So, (first) later assignment is still an initialization or not? In other words, initialization is just the process of giving a variable a first value, no matter this happens in its definition sentence or later in another sentence. ??



Alex
April 24, 2019 at 12:34 pm · Reply

No, the first assignment is not an initialization, it's an an assignment.

* Initialization = providing a value to an object at the point where it is defined
* Assignment = providing a value to an object at some point beyond the definition
* Uninitialized = An object that has not been given a known value

It's a bit confusing because we naturally expect the terms "initialized" and "uninitialized" to be opposites, but they're not! A variable that is assigned a known value is no longer uninitialized, but it may not have ever been initialized. I'll note this in the lesson.

alfonso
April 25, 2019 at 12:54 am · Reply

I get it now. Thank you!

```
1 int a; // no initialization, a is uninitialized
2 a = 7; // still no initialization but now a is no more uninitialized. a is not ini
```

Bella
June 1, 2019 at 6:53 am · Reply

How are they technically not opposites though?
Initialize is basically providing a value for a variable at the same time (After the variable is defined)
Uninitialized is basically a variable (Object) that hasn't been given a value.

One is providing value, whilst the other hasn't been given a value.
Or am I wrong?

Alex
June 1, 2019 at 10:56 am · Reply

A variable that was not initialized but was later given a value through assignment now has a known value. Thus, it is no longer considered "uninitialized", even though it was assigned a value, not initialized with one.

Mike
September 7, 2019 at 11:12 am · Reply

I'm glad I found this reply, it really cleared up some misunderstandings I had. You really should just copy and paste this reply verbatim to the section where you introduce variables.

BTW: I'm referring to your first reply, "No, the first assignment is not an initialization, it's an an assignment."

Athena
March 3, 2019 at 6:46 pm · Reply

code straight up wont run

Error C2220 warning treated as error - no 'object' file generated

Warning C4100 'x': unreferenced formal parameter

Edit: If i remove the "x" in "&x" the code now works. not sure if its a typo or just me being stupid and missing something :/

nascardriver
March 4, 2019 at 4:41 am · Reply

Hi!

Your compiler complains, because @x is never used.

You can remove the "x", making the parameter anonymous.

```
1 void doNothing(const int &)
```

You could add an attribute, indicating that @x is unused on purpose.

```
1 void doNothing(const int &x [[maybe_unused]])
```

Or you could disable C4100.

@Alex (Do you get highlights for mentions or do you read everything anyway?)

Hana
February 28, 2019 at 10:51 am · Reply

It always returns me the value of 0 when I don't assign a value to x in my IDE.

nascardriver
March 2, 2019 at 6:10 am · Reply

The value is undefined. Changes to your code that seem unrelated could change the number. Changes to you compiler could change the number. Changes to your system could change the number.