

Forum:[Hack Forums](#)

Thread: [void types & return statements](#)

D_MuLa - January 9, 2017, 4:18 pm

I have a function that prints the same output whether void is used or not, and i'm not using any return statements. So why does void exist if all you have to do is not make a return statement??

0xDEAD10CC - January 9, 2017, 4:19 pm

Uhh what? void exists to avoid having a function that returns something when it doesn't need to. Not sure what you're asking. If you omit void as the return from a function prototype, most compilers assume the return is 'int' btw, and in modern standards, it's non-standard compliant.

Your question doesn't make sense.

Annabelle Lee - January 9, 2017, 4:22 pm

The point of void functions is that you don't always need a value to return. For example you have an output function. What or why does it need to return anything? What about when you have a function to change some parameters or variables in the program internally, using a pointer. There are many other reasons, you just have to think a bit.

D_MuLa - January 9, 2017, 4:22 pm

(Jan 9, 2017 05:19 PM)0xDEAD10CC Wrote: [\[CITE\]](#) Uhh what? void exists to avoid having a function that returns something when it doesn't need to. Not sure what you're asking. If you omit void as the return from a function prototype, most compilers assume the return is 'int' btw, and in modern standards, it's non-standard compliant.

Your question doesn't make sense.but you can only return something by using the return statement, that's why i'm confused

Annabelle Lee - January 9, 2017, 4:23 pm

(Jan 9, 2017 05:22 PM)D_MuLa Wrote: [\[CITE\]](#) but you can only return something by using the return statement, that's why i'm confused
But what if you don't NEED to return anything?

D_MuLa - January 9, 2017, 4:28 pm

(Jan 9, 2017 05:23 PM)AnnabelleLee Wrote: [\[CITE\]](#) But what if you don't NEED to return anything? then you just don't use the return statement?

caiveman - January 9, 2017, 4:35 pm

This is right

```
Code void add(int a, int b){  
    printf("%d",a+b);  
    return;  
}
```

This is also right

```
Code void add(int a, int b){  
    printf("%d",a+b);
```

```
}
```

This is also right

```
Code int add(int a, int b){  
    int c = a+b;  
    printf("%d",c);  
    return c;  
}
```

This is not right (even though it might compile)

```
Code void add(int a, int b){  
    int c = a+b;  
    printf("%d",c);  
    return c;  
}
```

Annabelle Lee - January 9, 2017, 4:35 pm

(Jan 9, 2017 05:28 PM)D_MuLa Wrote: [\[CITE\]](#) then you just don't use the return statement? Because that is undefined behavior.

TouchMyMalware - January 9, 2017, 5:56 pm

You have to understand what's happening underneath and what's going on with the calling convention.

When you use a return type, the compiler ensures the return value goes into the eax register, depending,

on the calling convention you are using. I'll assume cdecl.

```
Code mov eax[esp+04h]
```

```
add eax, [esp+08h]
```

```
ret
```

This would be a simple add function.

The reason the compiler decided to use eax was because you specifically asked for the return type.

```
Code int result = add(5,10);
```

What's happening in the disassembly when you call it, is this.

```
Code push 0ah
```

```
push 05h
```

```
call add
```

```
add esp,08h
```

```
mov [ebp-04h],eax
```

Now let's assume you didn't ask for a return type.

Does the function still return a value? Of course, it's in the eax register;

however, if you didn't explicitly ask for a return type and used void the compiler can result in using any register.

```
Code mov edx,[esp+04h]
```

```
add edx,[esp+08h]
```

```
ret
```

Now what happens is completely up to the compiler.

You could still move eax into [ebp-4] however the value of eax,

is completely undefined. In fact the compiler in the latter function decided to use edx not eax.

So if you happen to get the same return value with void and int, it's completely coincidental. The compiler just decided to use eax in both instances for you.

0xDEAD10CC - January 9, 2017, 6:33 pm

(Jan 9, 2017 05:23 PM)AnnabelleLee Wrote: [\[CITE\]](#) But what if you don't NEED to return anything?

Then you use void and don't use a return statement (unless you want to return to the caller early without executing some other code based on some condition)? I think this is a pretty simple concept...

Annabelle Lee - January 9, 2017, 6:39 pm

(Jan 9, 2017 07:33 PM)0xDEAD10CC Wrote: [\[CITE\]](#) Then you use void and don't use a return statement (unless you want to return to the caller early without executing some other code based on some condition)? I think this is a pretty simple concept...

Ah.. I think there was a slight misunderstanding. I believe I was trying to show that if he had an int function and then didn't need to return anything, you should use a void instead, because having a type and not returning is undefined behavior. I believe the OP was just saying not to use return and it shouldn't have a point of having void function. That's how I understood thread, apologies if it was incorrect or I misread somewhere.

0xDEAD10CC - January 9, 2017, 7:27 pm

Ohh, for some reason I thought the OP typed that not you. I still don't understand what his issue is with return and void.

void if you don't need to return a value to the caller, and non-void if you do. Omit the return statement on void unless you are using it to prematurely return to the caller, otherwise return a value on all codepaths for a function that returns non-void.

Pretty much end of topic right there.

Kantos - January 9, 2017, 7:43 pm

TouchMyMalware stated one of the real reasons there's a void type, it helps with type safety.

Your non-void function is returning something as far as your compiler and machine code is concerned, but its value is undefined. Many languages analyze your code and throw a compile error if it detects that a function that expects a return value can exit without executing a return statement

If you're not using the return value of a function then it doesn't really matter if it's undefined. In pre-C89 standard C there wasn't necessarily a void type for functions, every function could return something and if a return type wasn't defined it was assumed to be int. From a code-readability standpoint a void type let's you know that this function does not have a return value and from a type-checking standpoint:

```
void somevoidfunc(){
//do something but return nothing
}
```

```
int someintfunc(){
//do something but return nothing, no compile error since not required by standard nor enforced by
all compiler authors
}
```

```
somefunc(){
//do something but return nothing, no void type required in pre-C89 so this is assumed to return int
and all functions could return something
}
```

```
int a = somevoidfunc(); //type mismatch, compile error
int b = someintfunc(); //no compile error, value of b is undefined
```

```
int c = somefunc(); //no compile error, value of c is undefined
```

Edit: posts above me clarified this as well

6577657534735 - January 10, 2017, 1:04 am

It's for type safety OP. C++ is a "strongly typed" language meaning types have to match up. You can't have anything untyped except by using auto which still requires the ability to deduce types from the context

g4143 - January 10, 2017, 5:15 am

(Jan 10, 2017 02:04 AM)Abraluna Wrote: [\[CITE\]](#) It's for type safety OP. C++ is a "strongly typed" language..

Well if you mean statically typed = strongly typed, then yes. Having some experience in strongly typed languages(languages which go further than statically typed = strongly typed) I find it hard to categorize C++ as strongly typed.

D_MuLa - January 10, 2017, 2:08 pm

thank you so much everyone lol void was really bothering me but i understand it completely now due to the various explanations given to me. @Kantos your post helped alot because you really dumbed it down for me which is what i needed

6577657534735 - January 10, 2017, 11:58 pm

(Jan 10, 2017 06:15 AM)g4143 Wrote: [\[CITE\]](#) Well if you mean statically typed = strongly typed, then yes. Having some experience in strongly typed languages(languages which go further than statically typed = strongly typed) I find it hard to categorize C++ as strongly typed.

Oops yes I meant statically not strongly