



## Why Lambda?

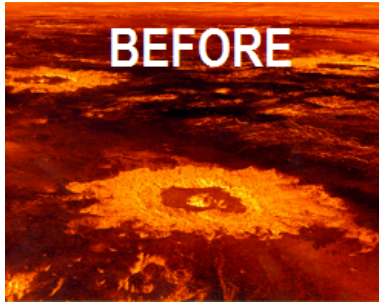
- Locally Declare Functions
- Pass functions as parameters (like functors)
- Use variables outside the scope of lambda
- Works well with STL

*I've started using lambda functions in production code  
... shortening code... improving unit tests... replacing  
what could previously have only been accomplished with  
macros.*<sup>1</sup>

---

Alex Allain

<http://www.cprogramming.com/c++11/c++11-lambda-closures.html>



## Why?

- Functors and similar operations cumbersome to use
- Overkill to declare and only use a function once <sup>a</sup>

---

*Stack Overflow Moderator, Flexo*

<sup>a</sup>

<http://stackoverflow.com/questions/7627098/what-is-a-lambda-expression-in-c11>

# Life Before Lambda Example

## Utilizing for-each without Lambda Functions

```
#include <algorithm>
#include <vector>

namespace {
    struct f {
        void operator()(int val) {
            // do something
        }
    };
}

void func(std::vector<int>& v) {
    f f;
    std::for_each(v.begin(), v.end(), f);
}
```

# Life Before Lambda Example

One might be tempted to use this code: <sup>2</sup>

Locally declared:

```
void func2(std::vector<int>& v) {  
    struct {  
        void operator()(int val) {  
            // do something  
        }  
    } f;  
    std::for_each(v.begin(), v.end(), f);  
}
```

However, our struct 'f' cannot be passed to a template function!

---

*Stack Overflow Moderator, Flexo*

<http://stackoverflow.com/questions/7627098/what-is-a-lambda-expression-in-c11>

*Use of lambdas: For ages, people have complained about having to write functions or (better) function objects for use as operations, such as Cmp above, for standard library (and other) algorithms. This was especially painful to do if you wrote large functions (don't) because in C++98 you could not define a local function object to use as an argument; now you can. However, lambdas allows us to define operations "inline:" - Bjarne Stroustrup<sup>3</sup>*

# Life With Lambda Example

Here is the previous example, resolved cleanly and effectively<sup>4</sup> thanks to the use of a lambda function.

## For\_each with Lambda

```
void func3(std::vector<int>& v) {  
    std::for_each(v.begin(), v.end(),  
        [](int val) { /* do something here*/ });  
}
```

---

*Stack Overflow Moderator, Flexo*

<http://stackoverflow.com/questions/7627098/what-is-a-lambda-expression-in-c11>



# The Community Responds

*Lambdas are syntactic sugar for functor classes . . . it aids in the readability of code (having your functor declared inline where it is used). - Terry Mahaffey*

*[In my opinion], the most important thing about lambda's is it keeps related code close together. - R Samuel Klatchko*

# The Community Responds

*There's no performance benefit per se, but the need for lambda came as a consequence of the wide adoption of the STL and its design ideas. - Fabio Ceconello*

*... is it just a neat development perk open for abuse by poor coders trying to look cool? - LoudNPossiblyWrong*

# Variable Capture

*They can capture contexts, and they do so by name and then they can take those contexts elsewhere and execute.*<sup>5</sup>

## Variable Capture Modes:

- [=] Set default capture mode to capture by value.  
(Data duplicated/Copy constructor invoked)
- [&] Set default capture mode to capture by reference.  
(No data duplication, function side-effects can persist)
- [variable\_name] Capture individual variables.  
(In this case, by value)
- [&variable\_name] Same as above, but by reference.
- [ ] The function body captures no variables.

Cubbi

# Variable Capture Example

## Capturing By-Value:

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

int main() {
    string input;
    cout << "enter a number: ";
    cin >> input;

    auto lambda_isNeg = [input] {
        return input < 0 ? "negative" : "positive"
    }

    cout << "you entered a " << lambda_isNeg;
    cout << "number" << endl;
    return 0;
}
```

# Lambda syntax

## Constructing a Lambda function

```
lambda_type my_func =  
    [capture_specification] (parameters) ->  
        return_type {function_body}
```

- The function name is not required, in the case of anonymous functions.
- The capture specification field can be left empty if desired.
- Parameters and return type are not necessary in most cases:
  - 1 Some compilers require a return type.
  - 2 Other compilers will default to void.

# Lambda syntax

## Example:

```
#include <iostream>

using namespace std;

int main()
{
    auto func = [] () { cout << "Hello world"; };
    func(); // now call the function
}
```