COEN-244 Tutorial #11

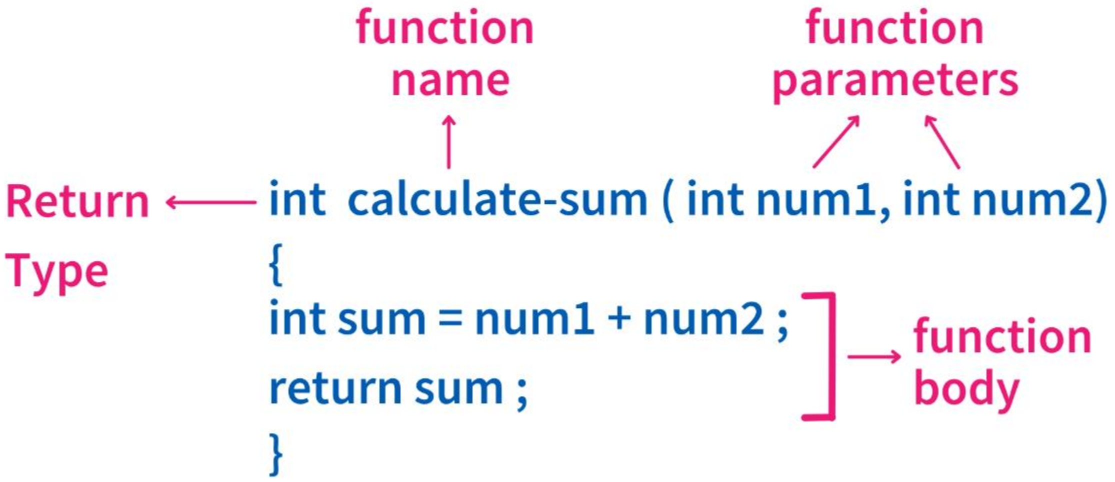
April 6th, 2023

Regular Functions

Functions: a set of statements gathered together to perform a specific task when the function is called.

- Body of the function is executed only when the function is called in `main()`
- It has a return-type and an *optional* parameter passing
- Functions are great tools for code reusability

<u>Code should always be broken into smaller, maintainable, and reusable chunks.</u>

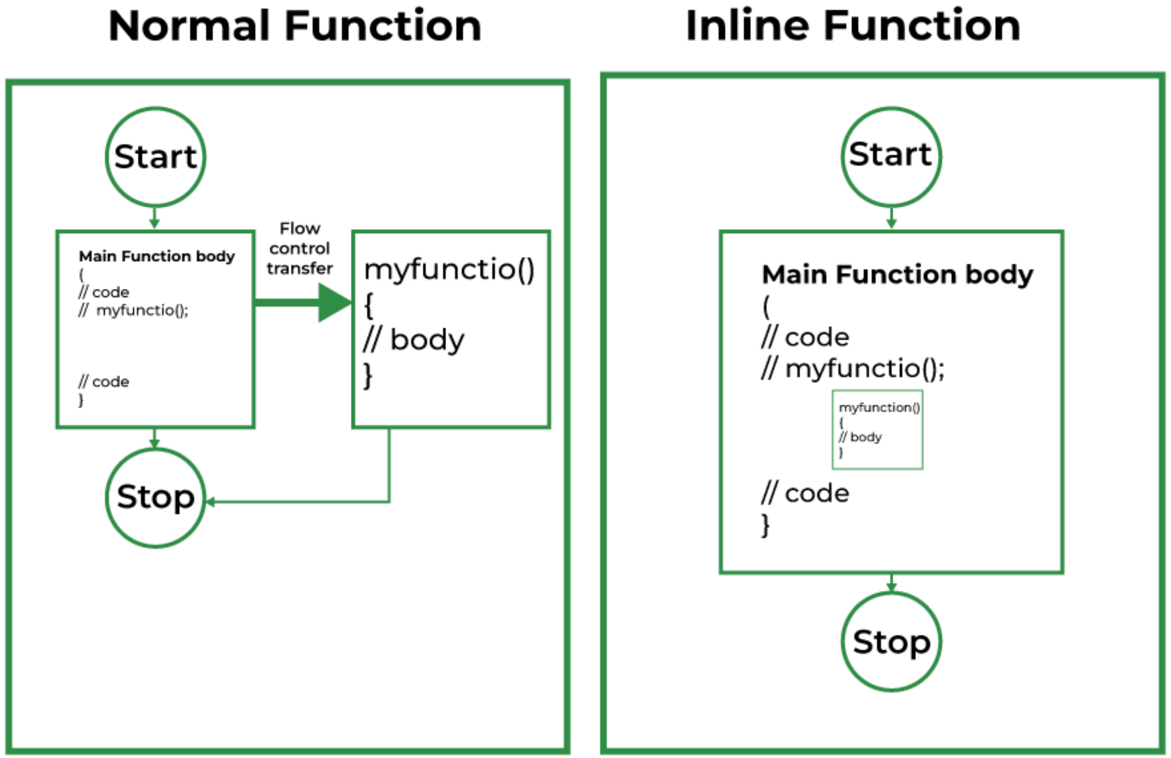


```
int  calculate-sum ( int num1, int num2)
{
int sum = num1 + num2 ;
return sum ;
}
```

Inline Functions

Inline Function: a function that is expanded in line when called.

- The whole body of the function gets inserted where it is called.
- No function call overhead, so it can increase efficiency



Normal Function

Inline Function

```
Main Function body
(
// code
//  myfunctio();


// code
}
```

```
myfunctio()
{
// body
}
```

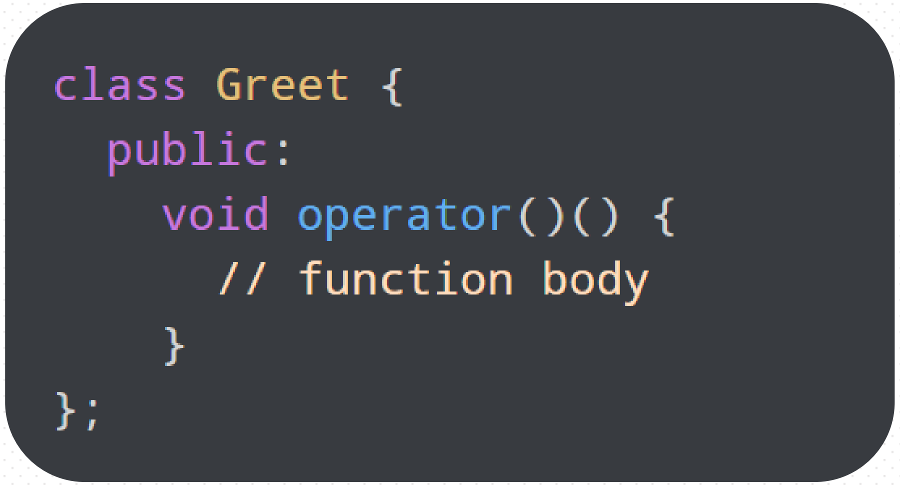
```
Main Function body
(
// code
// myfunctio();
    myfunction()
    {
    // body
    }
// code
}
```

Source: https://www.geeksforgeeks.org/inline-functions-cpp/

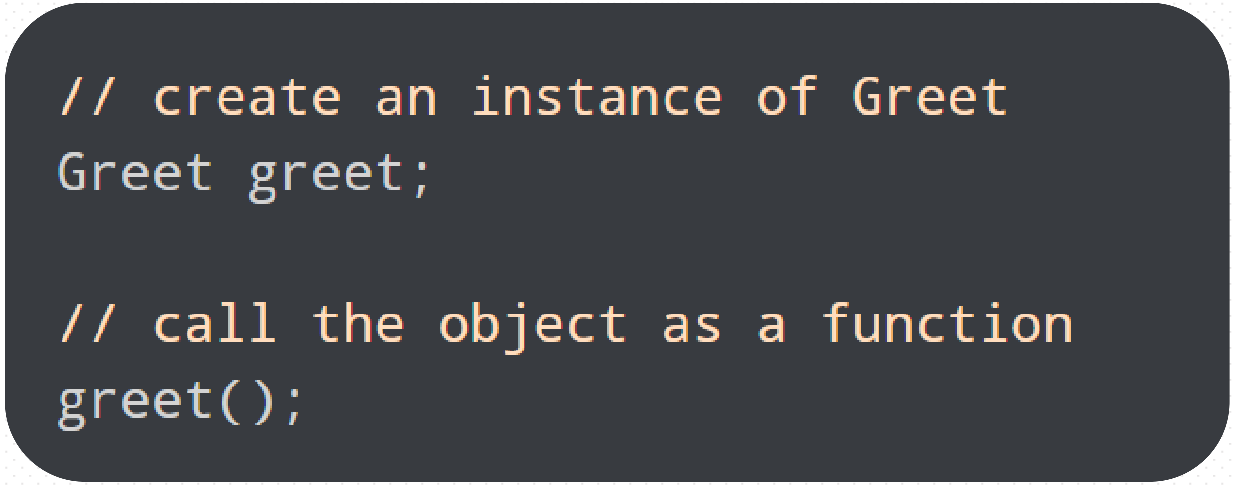
Functors

Functor (Function Object): a class object that can be called like a function.

- It is done by overloading the **`function-call operator ()`**
- Thus, we can have functions with more information
- This way we can empower the regular functions and perform operation on a basis of OOP



```
class Greet {
  public:
    void operator()() {
      // function body
    }
};
```

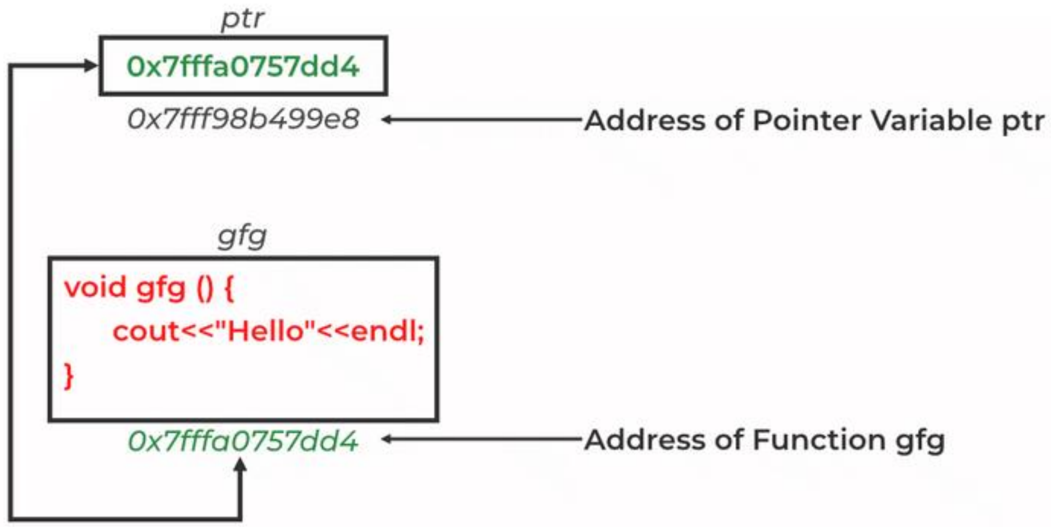


```
// create an instance of Greet
Greet greet;

// call the object as a function
greet();
```

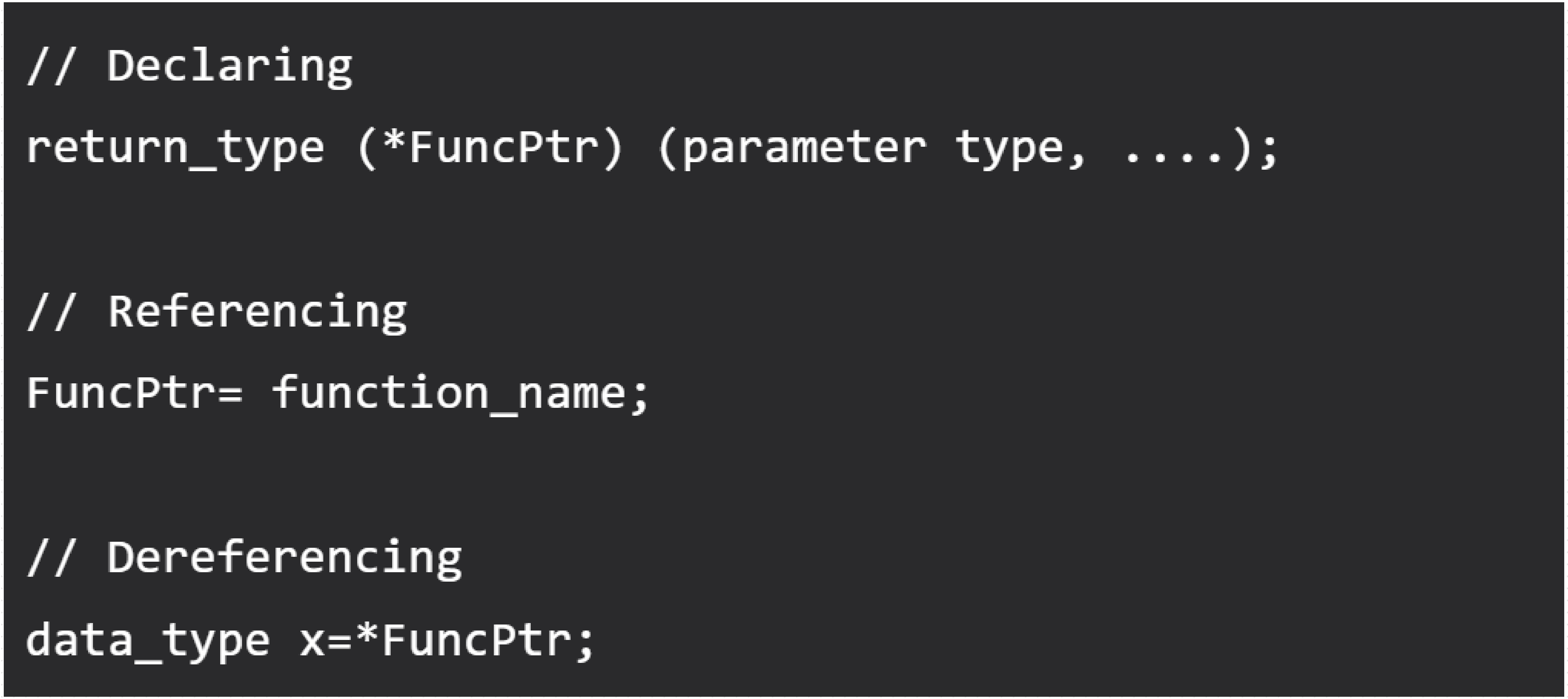
Pointers to Functions

- Since a function code resides in the memory, it also has an address
  - The address can be obtained by just writing the function name without '()'
- Hence we can have pointers to functions similar to objects.
  - This way, a function can be passed as a parameter to another function
  - This is different from a function returning a pointer



```
void gfg () {
    cout<<"Hello"<<endl;
}
```

SYNTAX: Pointers to Functions



```
// Declaring
return_type (*FuncPtr) (parameter type, ....);


// Referencing
FuncPtr= function_name;


// Dereferencing
data_type x=*FuncPtr;
```

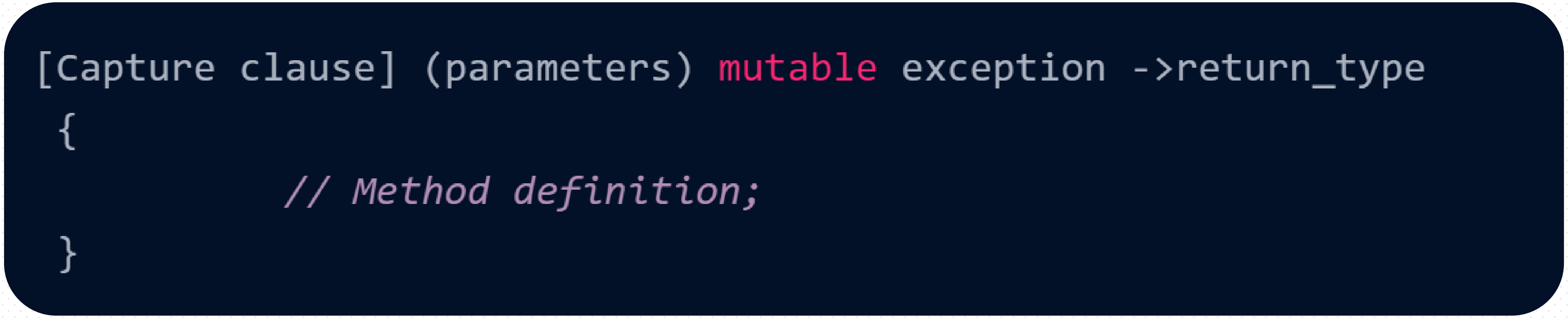
QUESTION: Why pointers to functions? How can they be useful? What could go wrong?

Lambda Expressions/Functions

Lambdas: powerful expressions that allows us to define anonymous functors which can be used inline or passed as an argument.

- It was introduced in C++11 to allow for short snippets of code with no name
- Lambdas can be very useful when we want to write fast and disposable functions.

Syntax:



```
[Capture clause] (parameters) mutable exception ->return_type
 {
           // Method definition;
 }
```

QUESTION: Why use Lambdas and when to use them?

Lambda Expressions/Functions

A Lambda (can) consists of:

- **Capture Clause** - a list of variables that are to be copied inside the lambda function in C++
- **Parameters** - zero, one or more than one argument to be passed to the lambda at execution time.
- **Mutable** - Mutual is an optional keyword. It lets us modify the value of the variables that are captured by the call-by-value when written in the lambda expression.
- **Return Type** - It is optional as the compiler evaluates it but in some complex cases compiler can't make out the return type and we need to specify it.
- **Body of the Method** - It is the same as the usual method definition.

THANK YOU