https://youtu.be/oTpGyu2L8C8

# C++11
# std::thread

https://youtu.be/oTpGyu2L8C8

# Topics

01 Advantages of std::thread

02 Understand std::thread

03 Examples of std::threads

# Part 01

## Advantages of std::thread

## Portable

C++ thread library makes your code platform independent.

Generic programming is boosted by using portable codes.

## Type safe

Allows passing multiple arguments to the thread handler in a type safe manner.

Compiler can ensure the right types and there by avoiding any sort of runtime issues.

## std::thread can be stack object

std::thread is mostly created as a stack object and hence it avoids all overheads of pointers, resource leaks etc.

It can also be created as pointer and can use with smart pointers which again eliminates any sort of pointer overheads

# Part 02

## Understand std::thread

# Understand std::thread

**Steps 1**

**Include the header file thread**

```
#include <thread>
```

This header file contains the implementations of class thread

# Understand std::thread

**Steps 2**

**Construct thread**

Constructs a thread object using on of the following of thread class

1. **Default constructor**
   - ➢ Construct a thread object that does not represent any thread of execution.
2. **Initialization constructor**
   - ➢ Construct a thread object that represents a new joinable thread of execution.
3. **Move constructor**
   - ➢ Construct a thread object from another thread. Transfers the ownerships. the thread that transferred ownership no longer represents any thread of execution.

https://youtu.be/oTpGyu2L8C8

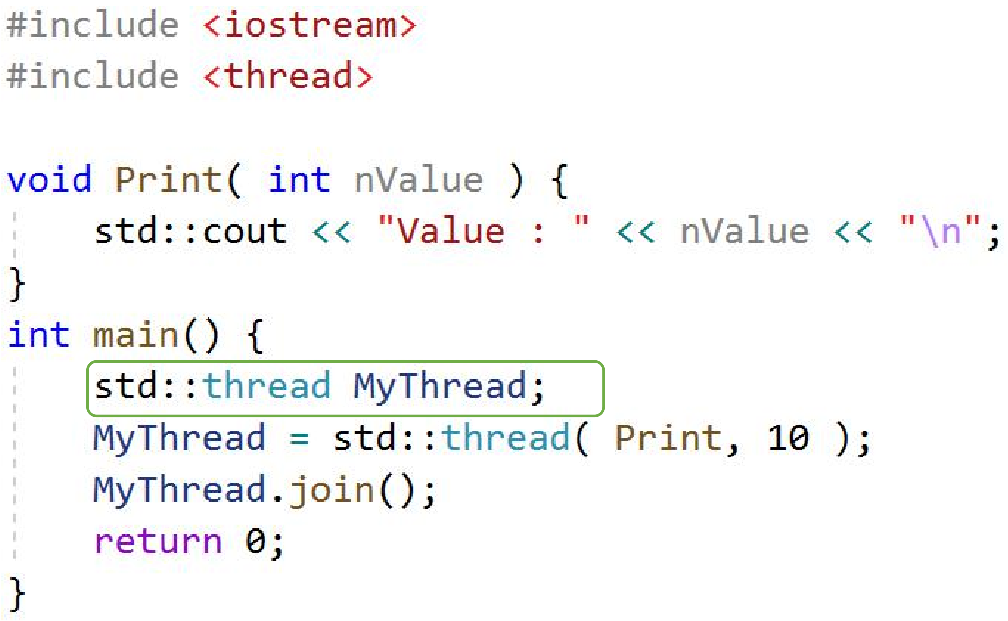
# Understand std::thread

### Steps 2

### Construct thread

### 1. Default constructor

- Constructs a thread object that **does not represent any thread of execution**.
- **Thread objects** created like this **are initialized later**.
  - This **allows** one **to set** the thread object **as a member varaible** etc.



```cpp
#include <iostream>
#include <thread>

void Print( int nValue ) {
    std::cout << "Value : " << nValue << "\n";
}
int main() {
    std::thread MyThread;
    MyThread = std::thread( Print, 10 );
    MyThread.join();
    return 0;
}
```

https://youtu.be/oTpGyu2L8C8

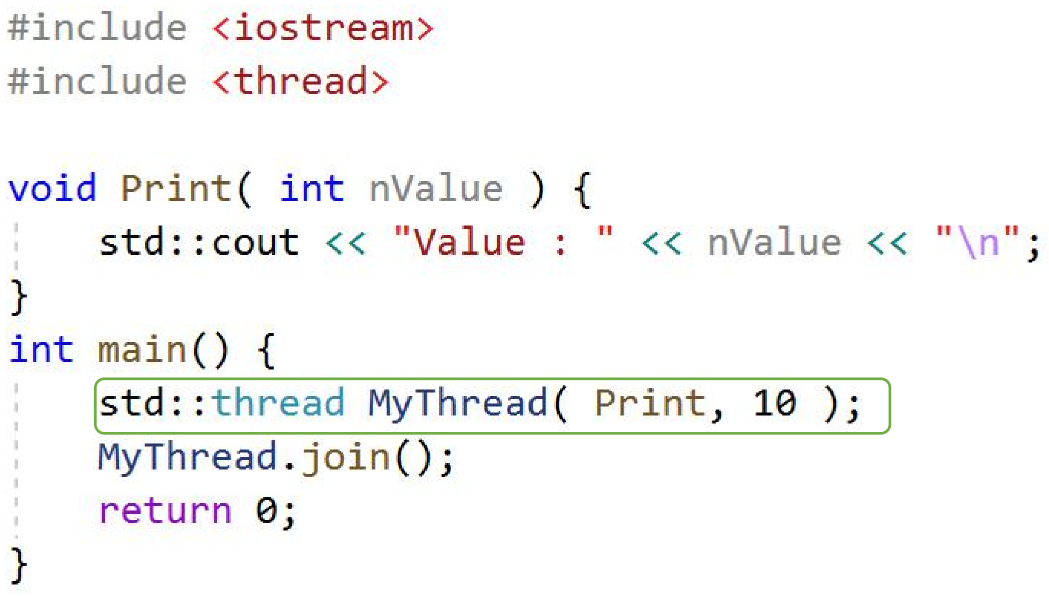
# Understand std::thread

## Steps 2

## Construct thread

### 2. Initialization constructor

- Construct a thread object that **represents a new joinable thread of execution**.
- The constructed thread starts executing immediately once the initialization construtor is executed.



```
#include <iostream>
#include <thread>

void Print( int nValue ) {
    std::cout << "Value : " << nValue << "\n";
}
int main() {
    std::thread MyThread( Print, 10 );
    MyThread.join();
    return 0;
}
```

https://youtu.be/oTpGyu2L8C8

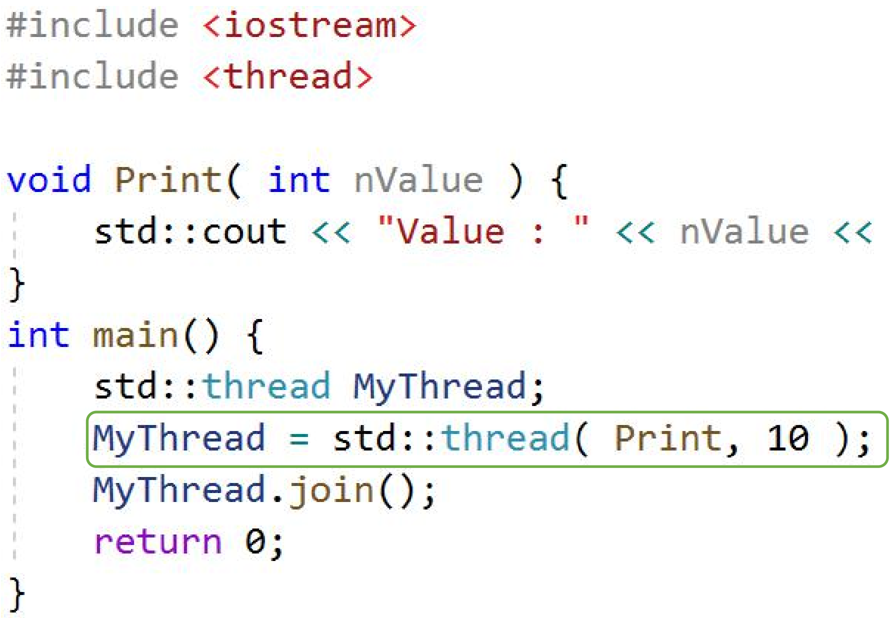
# Understand std::thread

## Steps 2

## Construct thread

### 3. Move constructor

- **Construct** a **thread** object **from another thread**.
- **Transfers** the **ownerships**.
- The thread that transferred ownership no longer represents any thread of execution.



```
#include <iostream>
#include <thread>

void Print( int nValue ) {
    std::cout << "Value : " << nValue << "\n";
}
int main() {
    std::thread MyThread;
    MyThread = std::thread( Print, 10 );
    MyThread.join();
    return 0;
}
```

https://youtu.be/oTpGyu2L8C8

# Understand std::thread

**Steps 3**

**Join/Detach thread**

Once the thread is constructed, either

- Wait for the thread to complete or
- Allow the thread to be free running

1. **Join Thread**
   - Blocks the current thread until the thread identified by *this finishes its execution
2. **Detach Thread**
   - Separates the thread of execution from the thread object, allowing execution to continue independently.

https://youtu.be/oTpGyu2L8C8

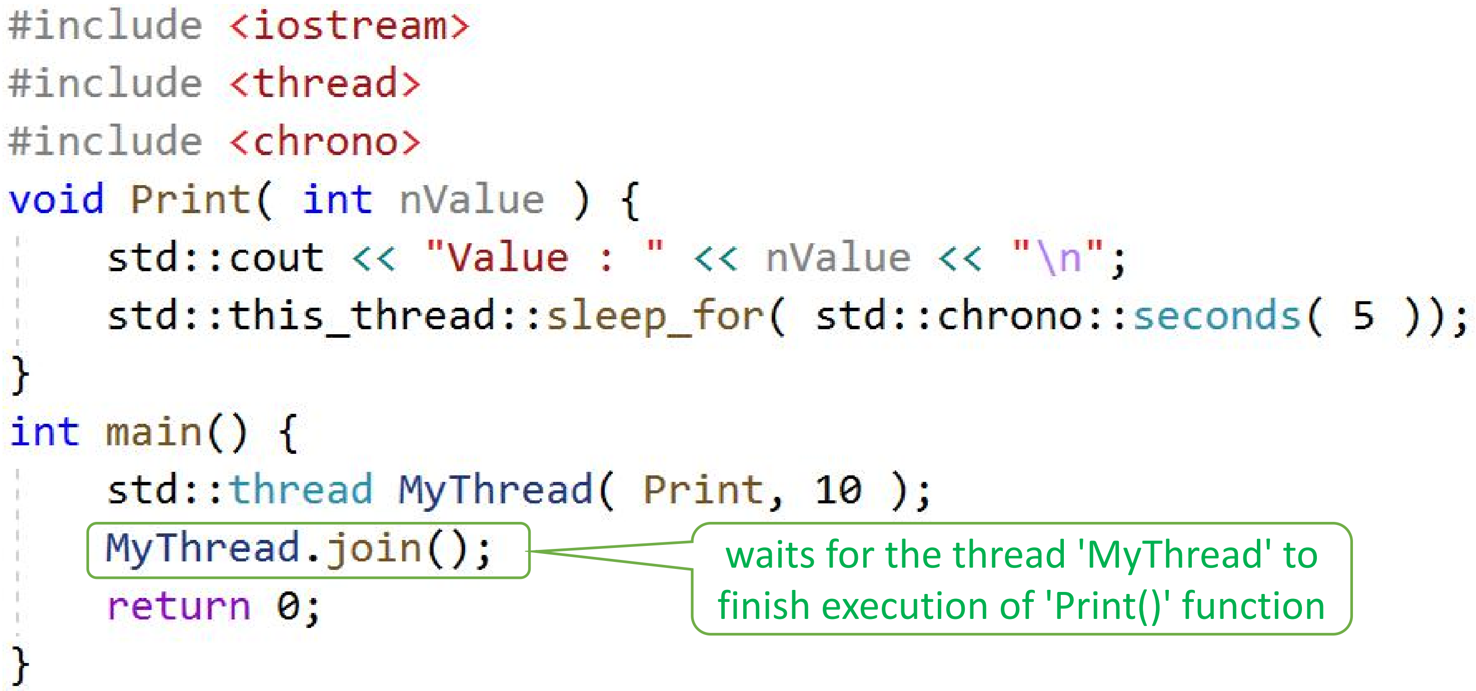
# Understand std::thread

## Steps 3

## Join/Detach thread

### 1. Join Thread

- Blocks the current thread until the thread identified by *this finishes its execution



```
#include <iostream>
#include <thread>
#include <chrono>
void Print( int nValue ) {
    std::cout << "Value : " << nValue << "\n";
    std::this_thread::sleep_for( std::chrono::seconds( 5 ));
}
int main() {
    std::thread MyThread( Print, 10 );
    MyThread.join();
    return 0;
}
```

waits for the thread 'MyThread' to finish execution of 'Print()' function

https://youtu.be/oTpGyu2L8C8

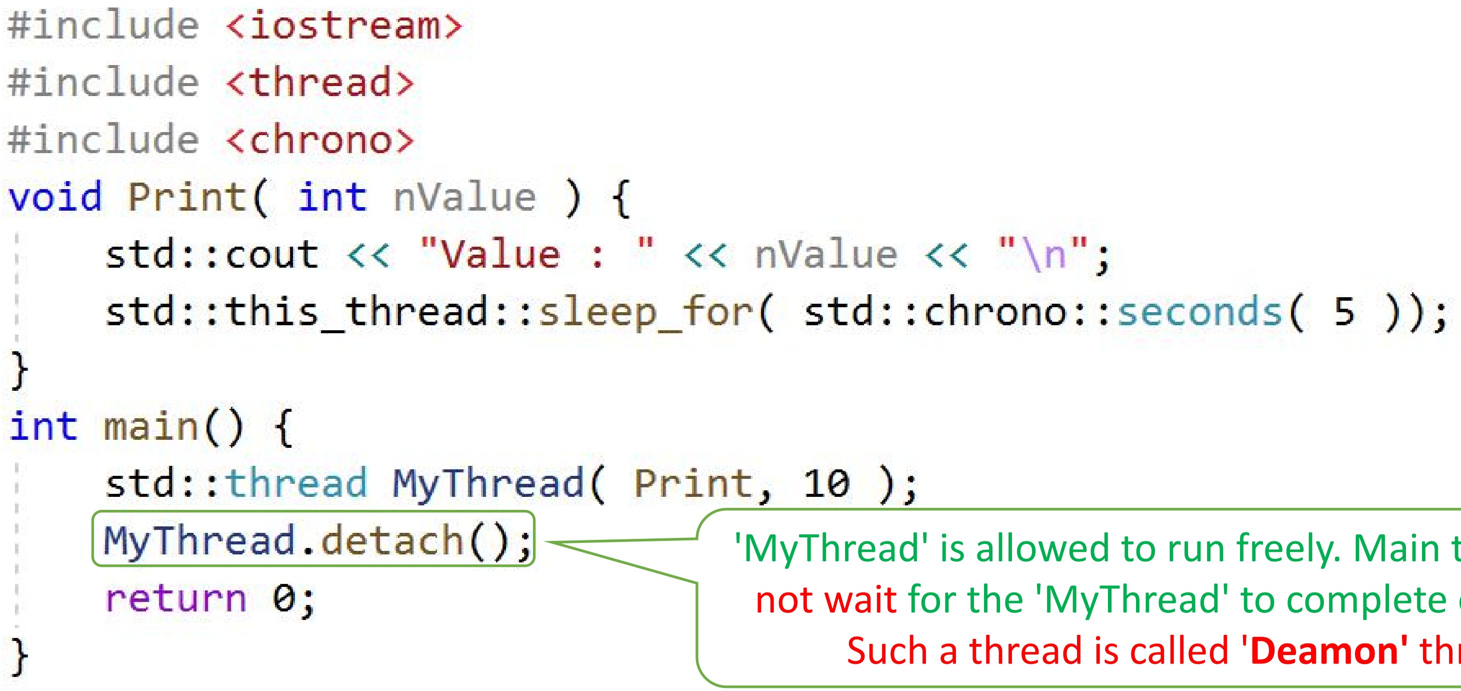
# Understand std::thread

**Steps 3**

**Join/Detach thread**

## 2. Detach Thread

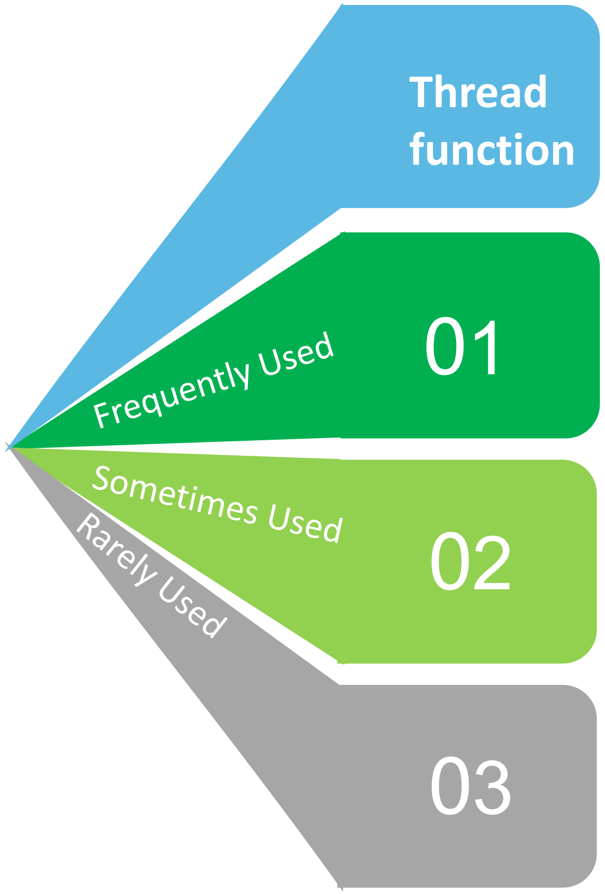
- ➢ Separates the thread of execution from the thread object, allowing independently execution



```
#include <iostream>
#include <thread>
#include <chrono>
void Print( int nValue ) {
    std::cout << "Value : " << nValue << "\n";
    std::this_thread::sleep_for( std::chrono::seconds( 5 ));
}
int main() {
    std::thread MyThread( Print, 10 );
    MyThread.detach();
    return 0;
}
```

'MyThread' is allowed to run freely. Main thread does not wait for the 'MyThread' to complete execution. Such a thread is called '**Deamon**' thread.

https://youtu.be/oTpGyu2L8C8



Thread function

Frequently Used 01

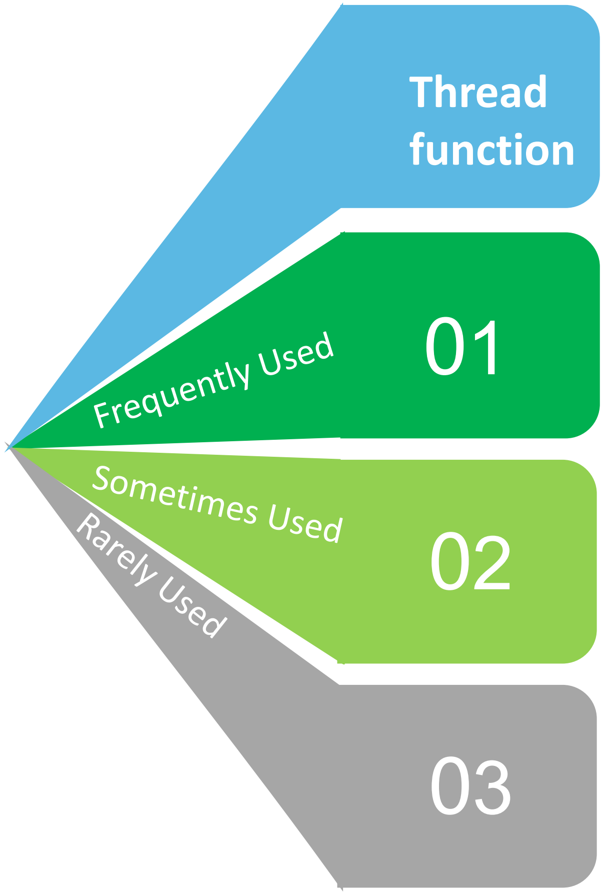
Sometimes Used 02

Rarely Used 03

**Thread function**

std::thread object uses one of the below option for specifying the thread function.

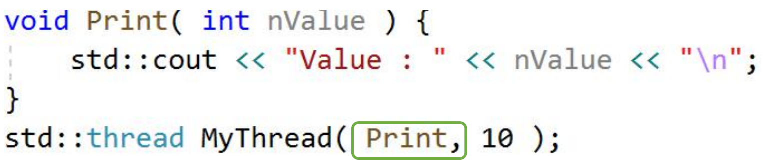
https://youtu.be/oTpGyu2L8C8



## Thread function

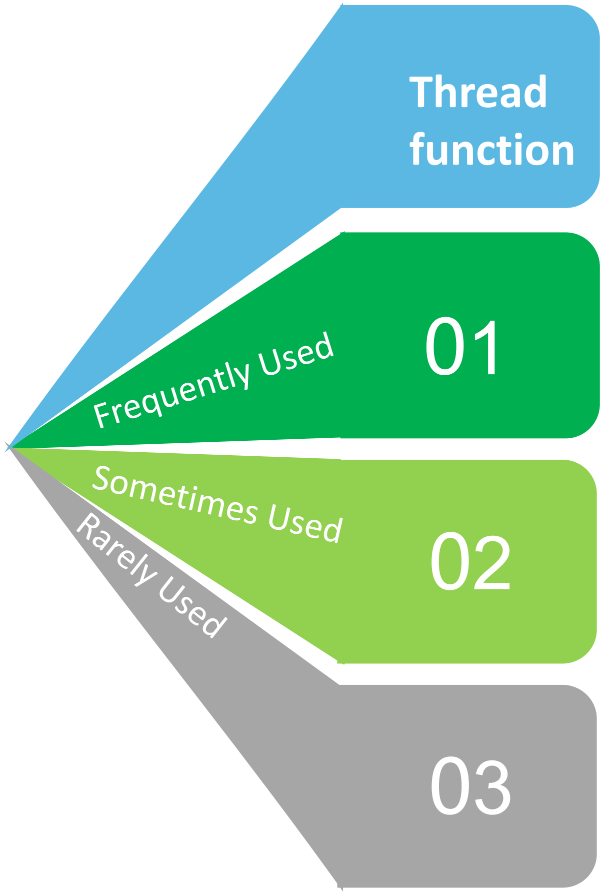
std::thread object uses one of the below option for specifying the thread function.

### Function Pointer



```
void Print( int nValue ) {
    std::cout << "Value : " << nValue << "\n";
}
std::thread MyThread( Print, 10 );
```

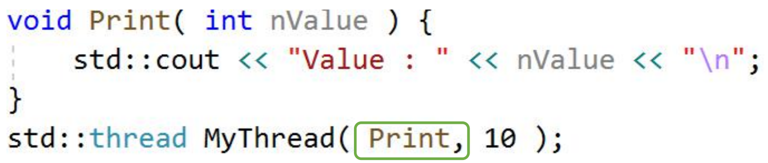
https://youtu.be/oTpGyu2L8C8



## Thread function

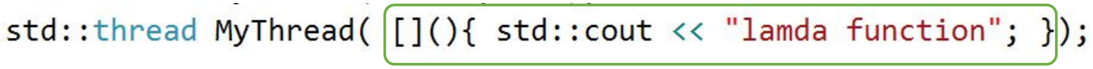
std::thread object uses one of the below option for specifying the thread function.

### Function Pointer



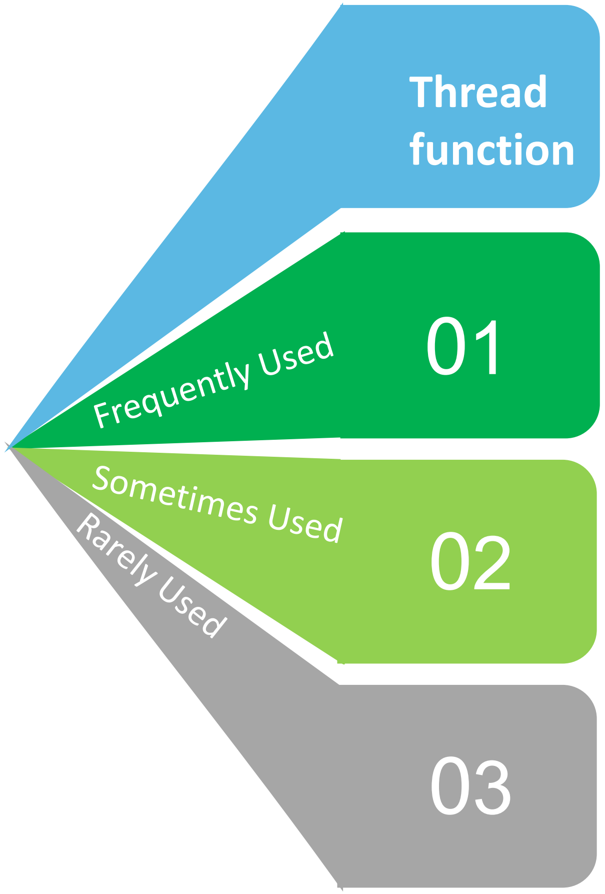
```
void Print( int nValue ) {
    std::cout << "Value : " << nValue << "\n";
}
std::thread MyThread( Print, 10 );
```

### Lamda Expresion



```
std::thread MyThread( [](){ std::cout << "lamda function"; });
```

https://youtu.be/oTpGyu2L8C8



Thread function

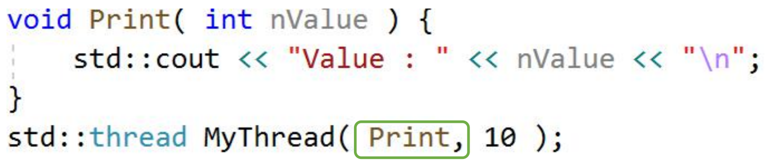
Frequently Used 01

Sometimes Used 02

Rarely Used 03

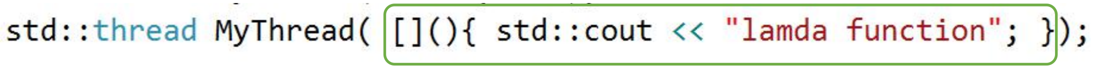
std::thread object uses one of the below option for specifying the thread function.

## Function Pointer



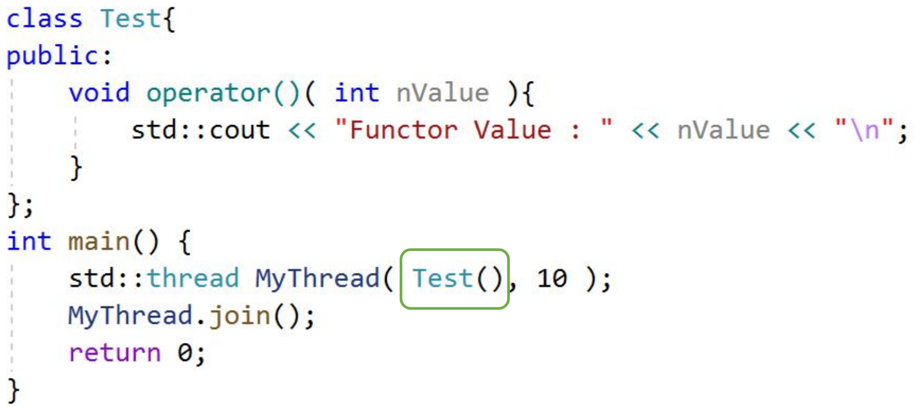
```
void Print( int nValue ) {
    std::cout << "Value : " << nValue << "\n";
}
std::thread MyThread( Print, 10 );
```

## Lamda Expresion



```
std::thread MyThread( [](){ std::cout << "lamda function"; });
```

## Functor



```
class Test{
public:
    void operator()( int nValue ){
        std::cout << "Functor Value : " << nValue << "\n";
    }
};
int main() {
    std::thread MyThread( Test(), 10 );
    MyThread.join();
    return 0;
}
```

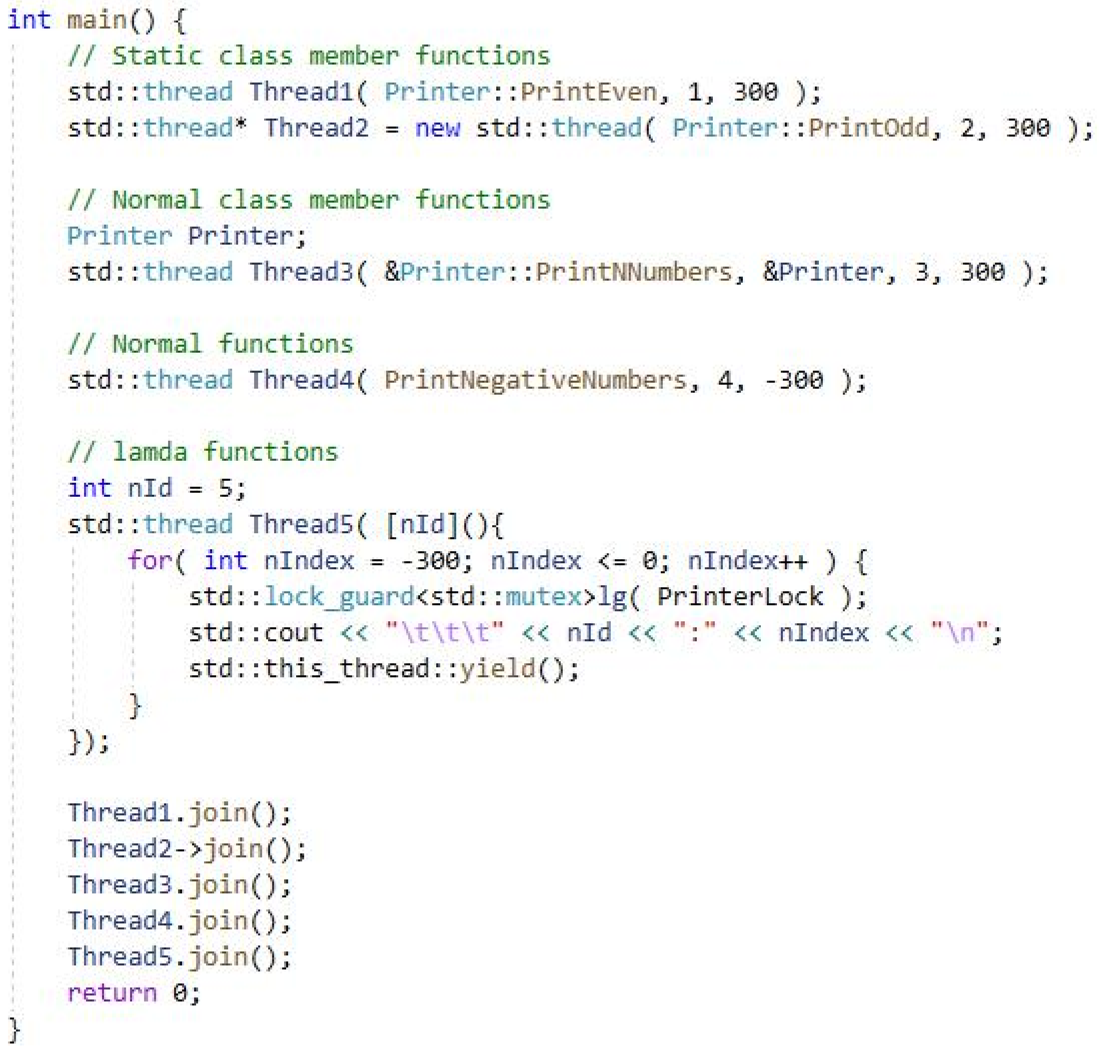
# Part 03

Examples of std::threads

https://youtu.be/oTpGyu2L8C8

# Examples of std::threads

https://youtu.be/oTpGyu2L8C8



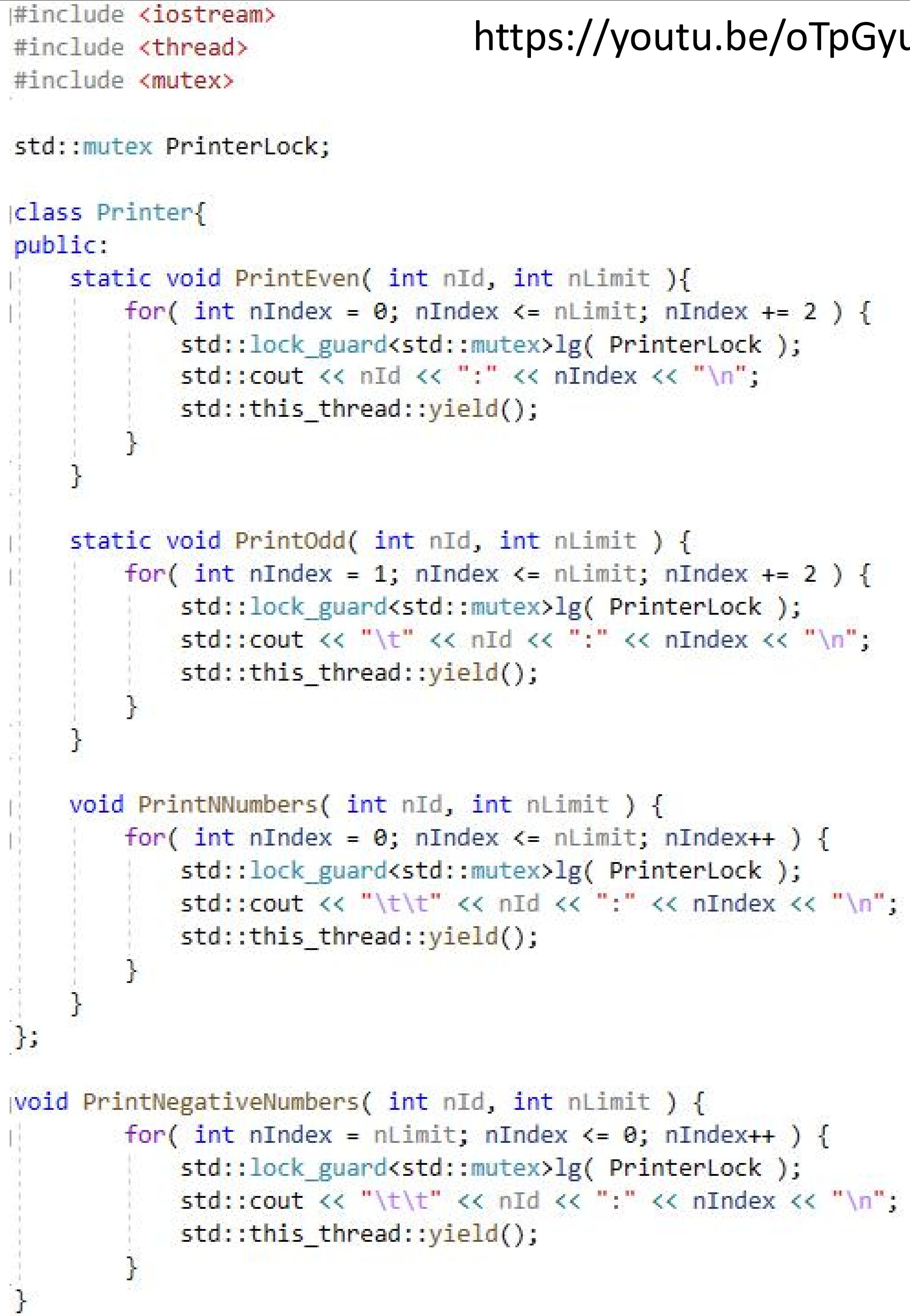
```cpp
int main() {
    // Static class member functions
    std::thread Thread1( Printer::PrintEven, 1, 300 );
    std::thread* Thread2 = new std::thread( Printer::PrintOdd, 2, 300 );

    // Normal class member functions
    Printer Printer;
    std::thread Thread3( &Printer::PrintNNumbers, &Printer, 3, 300 );

    // Normal functions
    std::thread Thread4( PrintNegativeNumbers, 4, -300 );

    // lamda functions
    int nId = 5;
    std::thread Thread5( [nId](){
        for( int nIndex = -300; nIndex <= 0; nIndex++ ) {
            std::lock_guard<std::mutex>lg( PrinterLock );
            std::cout << "\t\t\t" << nId << ":" << nIndex << "\n";
            std::this_thread::yield();
        }
    });

    Thread1.join();
    Thread2->join();
    Thread3.join();
    Thread4.join();
    Thread5.join();
    return 0;
}
```



```cpp
#include <iostream>
#include <thread>
#include <mutex>

std::mutex PrinterLock;

class Printer{
public:
    static void PrintEven( int nId, int nLimit ){
        for( int nIndex = 0; nIndex <= nLimit; nIndex += 2 ) {
            std::lock_guard<std::mutex>lg( PrinterLock );
            std::cout << nId << ":" << nIndex << "\n";
            std::this_thread::yield();
        }
    }

    static void PrintOdd( int nId, int nLimit ) {
        for( int nIndex = 1; nIndex <= nLimit; nIndex += 2 ) {
            std::lock_guard<std::mutex>lg( PrinterLock );
            std::cout << "\t" << nId << ":" << nIndex << "\n";
            std::this_thread::yield();
        }
    }

    void PrintNNumbers( int nId, int nLimit ) {
        for( int nIndex = 0; nIndex <= nLimit; nIndex++ ) {
            std::lock_guard<std::mutex>lg( PrinterLock );
            std::cout << "\t\t" << nId << ":" << nIndex << "\n";
            std::this_thread::yield();
        }
    }
};

void PrintNegativeNumbers( int nId, int nLimit ) {
        for( int nIndex = nLimit; nIndex <= 0; nIndex++ ) {
            std::lock_guard<std::mutex>lg( PrinterLock );
            std::cout << "\t\t" << nId << ":" << nIndex << "\n";
            std::this_thread::yield();
        }
}
```

# THANK YOU

LIKE & SHARE

&

Subscribe to our YouTube Channel

https://youtu.be/oTpGyu2L8C8