- Examples

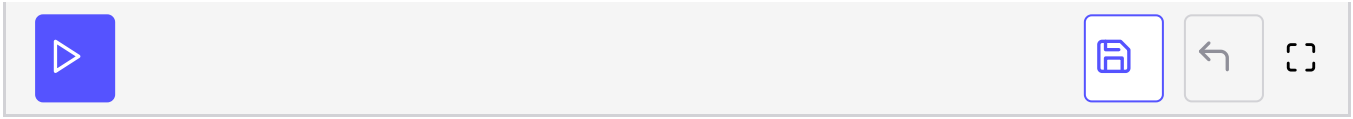
Let's take a look at examples for creating threads, thread lifetimes, and thread methods.

WE'LL COVER THE FOLLOWING

Example 1

```
#include <iostream> //threadCreate.cpp
#include <thread>

void helloFunction(){
  std::cout << "Hello C++11 from a function." << std::endl;
}

class HelloFunctionObject  {
  public:
    void operator()() const {
      std::cout << "Hello C++11 from a function object." << std::endl;
    }
};

int main(){

  std::cout << std::endl;

  // thread executing helloFunction
  std::thread t1(helloFunction);

  // thread executing helloFunctionObject
  HelloFunctionObject helloFunctionObject;
  std::thread t2(helloFunctionObject);

  // thread executing lambda function
  std::thread t3([]{std::cout << "Hello C++11 from lambda function." << std::endl;});
```

```
  // ensure that t1, t2 and t3 have finished before main thread terminates
  t1.join();
  t2.join();
  t3.join();

  std::cout << std::endl;

};
```



Explanation

- All three threads (`t1`, `t2`, and `t3`) write their messages to the console. The work package of thread `t2` is a function object (lines 8 - 13). The work package of thread `t3` is a lambda function (line 27).
- In lines 30 - 32, the main thread is waiting until its children are done.

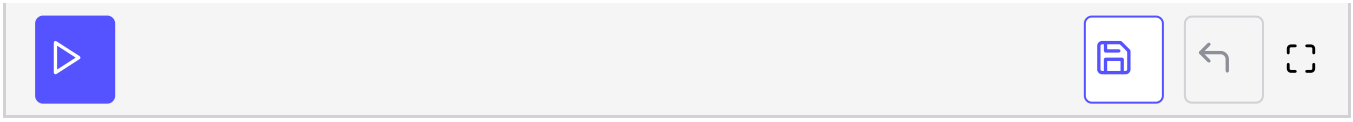
Let's take a look at the output.

- The three threads are executed in an arbitrary order. Even the three output operations can interleave.
- The creator of the child is responsible for the lifetime of the child. In this case, this occurred in the main thread.

Example 2

```
// threadLifetime.cpp
#include <iostream>
#include <thread>

void helloFunction(){
  std::cout << "Hello C++11 from a function." << std::endl;
}

class HelloFunctionObject  {
  public:
    void operator()() const {
      std::cout << "Hello C++11 from a function object." << std::endl;
    }
};

int main(){

  std::cout << std::endl;
```

```
  // thread executing helloFunction
  std::thread t1(helloFunction);


  // thread executing helloFunctionObject
  HelloFunctionObject helloFunctionObject;
  std::thread t2(helloFunctionObject);

  // thread executing lambda function
  std::thread t3([]{std::cout << "Hello C++11 from lambda function." << std::endl;});

  // ensure that t1, t2 and t3 have finished before main thread terminates
  t1.join();
  t2.detach();
  t3.join();

  std::cout << std::endl;

};
```



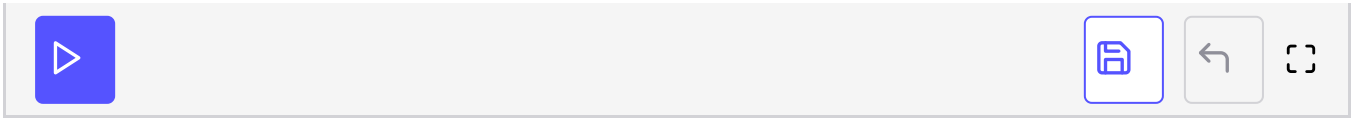
Explanation

- This program has a race condition.
- The thread `t2` is detached, meaning that the function `helloFunctionObject` cannot be performed if thread `t1` and `t2` are faster than thread `t3`.

Example 3

```
// threadMethods.cpp

#include <iostream>
#include <thread>

using namespace std;

int main(){

  cout << boolalpha << endl;

  cout << "hardware_concurrency()= "<< thread::hardware_concurrency() << endl;

  thread t1([]{cout << "t1 with id= " << this_thread::get_id() << endl;});
  thread t2([]{cout << "t2 with id= " << this_thread::get_id() << endl;});

  cout << endl;

  cout << "FROM MAIN: id of t1 " << t1.get_id() << endl;
  cout << "FROM MAIN: id of t2 " << t2.get_id() << endl;

  cout << endl;
```

```
  swap(t1,t2);

  cout << "FROM MAIN: id of t1 " << t1.get_id() << endl;
  cout << "FROM MAIN: id of t2 " << t2.get_id() << endl;

  cout << endl;

  cout << "FROM MAIN: id of main= " << this_thread::get_id() << endl;

  cout << endl;

  cout << "t1.joinable(): " << t1.joinable() << endl;

  cout << endl;

  t1.join();
  t2.join();

  cout << endl;

  cout << "t1.joinable(): " << t1.joinable() << endl;

  cout << endl;

}
```



Explanation

- In combination with the output, the program should be easy to follow.
- Maybe it looks a little weird that threads `t1` and `t2` (lines 14 and 15) run at different points in time of the program execution.
- You have no guarantee when each thread runs. It is only guaranteed that both threads will run before `t1.join()` and `t2.join()` in lines 38 and 39.
- The more mutable (non-const) variables the threads share, the more challenging multithreading becomes.

Level up your understanding of threads with a few exercises in the next lesson.