// threadWithoutJoin.cpp  
  
#include <iostream>

#include <thread>

int main(){

std::thread t([]{std::cout << std::this\_thread::get\_id() << std::endl;});

}

join and detach

The lifetime of the created thread t ends with its callable unit. The creator has two choices. First: it waits, until its child is done (t.join()). Second: it detaches itself from its child: t.detach(). A thread t with callable unit (you can create threads without callable unit) is joinable, in case there were no t.join() or t.detach calls to the thread. A joinable thread  destructor throws  std::terminate  exception. Thus, the program terminates. That is the reason, the actual run terminated unexpectedly.

The solution for this problem is simple. By calling t.join(), the program behaves as it should.

// threadWithJoin.cpp  
  
#include <iostream>

#include <thread>

int main(){

std::thread t([]{std::cout << std::this\_thread::get\_id() << std::endl;});

t.join();

}

Moving threads

It is not possible to copy a thread (copy semantic), you can only move ([move semantic](http://thbecker.net/articles/rvalue_references/section_02.html)) it.

// threadMoved.cpp  
  
#include <iostream>

#include <thread>

#include <utility>

int main(){

std::thread t([]{std::cout << std::this\_thread::get\_id();});

std::thread t2([]{std::cout << std::this\_thread::get\_id();});

t= std::move(t2);

t.join();

t2.join();

}

What is going wrong? We have two issues:

1. By moving (taking ownership of)  the thread t2, t gets a new callable unit and its destructor will be called. So t's destructor calls std::terminate, because it is still joinable.
2. Thread t2 has no associated callable unit. The invocation of join on a thread without callable unit leads to the exception std::system\_error.
3. // threadMovedFixed.cpp  
     
   #include <iostream>
4. #include <thread>
5. #include <utility>
6. int main(){
7. std::thread t([]{std::cout << std::this\_thread::get\_id() << std::endl;});
8. std::thread t2([]{std::cout << std::this\_thread::get\_id() << std::endl;});
10. t.join();
11. t= std::move(t2);
12. t.join();
14. std::cout << "\n";
15. std::cout << std::boolalpha << "t2.joinable(): " << t2.joinable() << std::endl;
16. }

## scoped\_thread

// scoped\_thread.cpp  
  
#include <iostream>

#include <thread>

#include <utility>

class scoped\_thread{

std::thread t;

public:

explicit scoped\_thread(std::thread t\_): t(std::move(t\_)){

if ( !t.joinable()) throw std::logic\_error("No thread");

}

~scoped\_thread(){

t.join();

}

scoped\_thread(scoped\_thread&)= delete;

scoped\_thread& operator=(scoped\_thread const &)= delete;

};

int main(){

scoped\_thread t(std::thread([]{std::cout << std::this\_thread::get\_id() << std::endl;}));

}