```
#include <memory>
#include <string>
#include <iostream>
#include <functional>
#include <queue>


template<typename Impl,  typename ...Args > struct Executor{

  void operator() (Args ...args){if(m_impl)  return (*m_impl)(args...); }
  template<typename ...IArgs > static Executor Instance(IArgs ...iargs){
      return Executor(std::make_shared<Impl>(iargs ...));
  }
  private:
  using ImplPtr = std::shared_ptr<Impl>;
  Executor(ImplPtr impl):m_impl(impl){}
  ImplPtr m_impl;
};

template<typename Impl>
using VoidExecutor = Executor<Impl> ;

struct I1{
    void operator() ( ) {std::cout << "I1::operator() "<<std::endl;}
};
struct I2{
    void operator() ( ) {std::cout << "I2::operator() "<<std::endl;}
};

struct I3{
    I3(int i):m_i(i){}
    void operator() () {std::cout << "I2::operator() "<< m_i<<std::endl;}
    int m_i;
};

std::queue< std::function<void()> > voidQueue;

int main(int argc, char* argv[])
{

     voidQueue.push(VoidExecutor<I1>::Instance());
     voidQueue.push(VoidExecutor<I2>::Instance());
     voidQueue.push(VoidExecutor<I3>::Instance(23));
     voidQueue.push([]() {std::cout << "lambda []() "<<std::endl;} );
     int j =5;
     voidQueue.push([j]() {std::cout << "lambda [](j) "<< j<<std::endl;} );
     voidQueue.push(I1());
     voidQueue.push(I3(8));

     while(!voidQueue.empty()){
         voidQueue.front()(); voidQueue.pop();
     }

}
```