1)std::bad_alloc
2)std::bad_cast
3)std::bad_exception
4)std::bad_typeid
5)std::logic_error
    5-1)std::domain_error: exception thrown when a mathematically invalid domain is used.
    5-2)std::invalid_argument
    5-3)std::length_error
    5-4)std::out_of_range
6)std::runtime_error: An exception that theoretically cannot be detected by reading the code.
    6-1)std::overflow_error :
  The only standard library components that
  throw std::overflow_error are std::bitset::to_ulong and std::bitset::to_ullong.
    6-2)std::underflow_error
    6-3)std::range_error

```
struct Foo { virtual ~Foo() {} };
struct Bar { virtual ~Bar() {} };


void bad_allocExample()
{
  try
  {
    while (true)
    {
      new int[100000000ul];
    }
  } catch (const std::bad_alloc& e)
  {
    std::cout << "Allocation failed: " << e.what() << '\n';
  }
}


void bad_castExample()
{
  Bar b;
  try
  {
    Foo& f = dynamic_cast<Foo&>(b);
  } catch(const std::bad_cast& e)
  {
    std::cout << e.what() << '\n';
  }
}


void bad_typeidExample()
{
  Foo* p = nullptr;
  try
  {
    std::cout << typeid(*p).name() << '\n';
  } catch(const std::bad_typeid& e) {
    std::cout << e.what() << '\n';
  }
}
```

```
void logical_ErrorExample()
{
   int amount, available;
   amount=10;
   available=9;
   if(amount>available)
   {
      throw std::logic_error("Error");
   }
   catch ( std::exception &e )
   {
    std::cerr << "Caught: " << e.what( ) << std::endl;
    std::cerr << "Type: " << typeid( e ).name( ) << std::endl;
   };
}

void domain_errorExample()
{
   try
   {
      const double x = std::acos(2.0);
      std::cout << x << '\n';
   }
   catch (std::domain_error& e)
   {
      std::cout << e.what() << '\n';
   }
   catch (...)
   {
      std::cout << "Something unexpected happened" << '\n';
   }

}

void invalid_argumentExample()
{
   try
   {
   // binary wrongly represented by char X
      std::bitset<32> bitset(std::string("0101001X01010110000"));
   }
   catch (std::exception &err)
   {
      std::cerr<<"Caught "<<err.what()<<std::endl;
      std::cerr<<"Type "<<typeid(err).name()<<std::endl;
   }
}

void length_errorExample()
{
   try
   {
      // vector throws a length_error if resized above max_size
      std::vector<int> myvector;
      myvector.resize(myvector.max_size()+1);
   }
   catch (const std::length_error& le)
   {
      std::cerr << "Length error: " << le.what() << '\n';
   }
}
```

```
void out_of_rangeExample()
{
   std::vector<int> myvector(10);
   try
   {
      myvector.at(20)=100;      // vector::at throws an out-of-range
   }
   catch (const std::out_of_range& oor)
   {
      std::cerr << "Out of Range error: " << oor.what() << '\n';
   }
}

void overflow_errorExample()
{
   try
   {
      std::bitset<100> bitset;
      bitset[99] = 1;
      bitset[0] = 1;
      // to_ulong(), converts a bitset object to the integer that would generate the sequence of bits
      unsigned long Test = bitset.to_ulong();
   }
   catch(std::exception &err)
   {
      std::cerr<<"Caught "<<err.what()<<std::endl;
      std::cerr<<"Type "<<typeid(err).name()<<std::endl;
   }
}


void range_errorExample()
{
   try
   {
     throw std::range_error( "The range is in error!" );
   }
   catch (std::range_error &e)
   {
     std::cerr << "Caught: " << e.what( ) << std::endl;
     std::cerr << "Type: " << typeid( e ).name( ) << std::endl;
   }
}


//Defining your exceptions

struct CustomException : public std::exception
{
  const char * what () const throw ()
  {
    return "CustomException happened";
  }
};

void customExceptionExample()
{
   try
   {
      throw CustomException();
```

```
    } catch(CustomException& e)
    {
       std::cout << "CustomException caught" << std::endl;
       std::cout << e.what() << std::endl;
    } catch(std::exception& e)
    {
    //Other errors
    }
}
```