Course 1-2 : class basics

main.cpp

```cpp
#include <stdio.h>
#include <iostream>

// int main()
// {
//   printf("Hello world\n");
//   return 0;
// }

class compteur {
public:
  int cpt;

  compteur() : cpt(0) {
    printf("%s called: cpt: %d, size: %zu\n", __PRETTY_FUNCTION__, cpt, sizeof(*this));
  }

  int inc() {
    printf("    compteur() +1 inc\n");
    cpt++;
    return cpt;
  };
};

class pairCompteur : public compteur {
protected:
    int cpt;
public:
  pairCompteur() : cpt(1) {
    printf("  %s called: cpt: %d, size: %zu\n", __PRETTY_FUNCTION__, cpt, sizeof(*this));
  }

  int inc() {
    printf("  pairCompteur() +2 inc\n");
    compteur::inc();
    reinterpret_cast<compteur &>(*this).inc();
    return cpt;
  };
};

class doublepairCompteur : protected pairCompteur {
  int cpt;
public:
  // pairCompteur::cpt; // conflict with above, but allows to change visibility

  doublepairCompteur() : cpt(2) {
    printf("    %s called: cpt: %d, size: %zu\n", __PRETTY_FUNCTION__, cpt, sizeof(*this));
  }

  int inc() {
```

```
    printf("doublepairCompteur() +4 inc\n");
    pairCompteur::inc();
    pairCompteur::inc();
    return cpt;
  };
  int inc2() {
    printf("doublepairCompteur() +8 inc\n");
    pairCompteur::inc();
    pairCompteur::inc();
    pairCompteur::inc();
    reinterpret_cast<pairCompteur &>(*this).inc(); // Dirty, but works too
    return cpt;
  };
  // int getCpt() const { return cpt; };
  // int getCpt() const { return this->cpt; };
  // int getCpt() const { return doublepairCompteur::cpt; };
  // int getCpt() const { return pairCompteur::cpt; };
  int getCpt() const { return compteur::cpt; };
};

typedef  int (doublepairCompteur::* t_doubleInc)(void); // Pointer on member function

int main()
{
  unsigned char c = 255;
  int i = c;

  doublepairCompteur cpt;

  t_doubleInc inc = &doublepairCompteur::inc2;

  printf("i == %d\n", i);

  cpt.inc();
  // std::cout << "cpt value: " << cpt.cpt << std::endl; // compile error with doublepair class
  std::cout << "cpt value: " << cpt.getCpt() << std::endl;

  (cpt).inc();
  // std::cout << "cpt value: " << cpt.cpt << std::endl; // compile error with doublepair class
  std::cout << "cpt value: " << cpt.getCpt() << std::endl;

  (cpt.*inc)(); // Same as above but with function pointer
  std::cout << "cpt value: " << cpt.getCpt() << std::endl;

  (cpt.*inc)();
  std::cout << "cpt value: " << cpt.getCpt() << std::endl;
}
```

Resources

- **Constructor and destructors**: https://isocpp.org/wiki/faq/ctors
- **Special members / default class methods**: https://cplusplus.com/doc/tutorial/classes2/
- **Copy constructors**: https://en.cppreference.com/w/cpp/language/copy_constructor
- **Move operator**: https://en.cppreference.com/w/cpp/language/move_assignment

- **Implicitly declared Ctors/Dtors**: https://en.cppreference.com/w/cpp/language/default_constructor

Course 3 : Templates

Source files

main.cpp

```
#include <iostream>
#include <cstdbool>
#include <cstdio>
#include "concatDigits.hh"

// See https://accu.org/journals/overload/9/43/frogley_442/
template<class valueT> // Custom traits / properties
struct numeric_traits
{
public:
  bool hasFractionalParts = false;
  template<class T>
  static valueT append(valueT aValue, T aDigit)
  {
    return aValue * 10 + aDigit;
  }
};

template<> struct numeric_traits<float>
{
public:
  bool hasFractionalParts = true;
  template<class T>
  static float append(float aValue, T aDigit)
  {
    char buffer[20];
    int len = snprintf(buffer, sizeof(buffer), "%f", aValue);
    int lastDigitPos = 0;

    for (auto i = len - 1 ; i > 0 and buffer[i] == '0' ; --i)
      lastDigitPos = i;
    buffer[lastDigitPos] = aDigit + '0';
    buffer[lastDigitPos + 1] = '\0';

    printf("%s: aValue = %f, aDigit = %f, buffer = %s\n", __PRETTY_FUNCTION__, aValue, aDigit, buffer);
    return atof(buffer);
  }
};

template<typename T1, typename T2, typename traits = numeric_traits<T1>>
T1 concatDigitsFullGeneric(T1 aValue, T2 aDigit, traits* = nullptr)
{
  if(aDigit < 0 || aDigit > 10)
    throw std::overflow_error("Is not a digit");
  return traits::append(aValue, aDigit);
}

int main()
```

```
{
  int val = 123;
  int newDigit = 5;

  printf("%d + %d -> %d\n",
     val, newDigit,
     concatDigits(val, newDigit));
  printf("%d + %d -> %d\n",
     val, newDigit,
     concatDigitsGeneric(val, newDigit));

  float fval = 12.34;
  float newfDigit = 5;
  printf("%f + %d -> %f\n", fval, newDigit, concatDigitsGeneric(fval, newfDigit));
  printf("%f + %d -> %f\n", fval, newDigit, concatDigitsFullGeneric(fval, newfDigit));

  return 0;
}
```

concatDigits.cpp

```cpp
#include <stdexcept>
#include "concatDigits.hh"

int concatDigits(int nb, int newDigit) {
  if (!(0 <= newDigit && newDigit <= 9))
    throw std::overflow_error("Is not a digit");
  return nb * 10 + newDigit % 10;
}

size_t concatDigits(size_t nb, size_t newDigit) {
  if (!(newDigit <= 9))
    throw std::overflow_error("Is not a digit");
  return nb * 10 + newDigit % 10;
}

template <typename T> T concatDigits(T nb, T newDigit) {
  if (!(0 <= newDigit && newDigit <= 9))
    throw std::overflow_error("Is not a digit");
  return nb * 10 + newDigit % 10;
}

// GENERIC VERSION: char, short, int, size_t, float, etc
template <typename T> T concatDigitsGeneric(T nb, T newDigit) {
  return nb * 10 + newDigit % 10; // Doesn't work with floats (modulo is not supported)
}

// float (specialization, for "float", use this one instead of the generic)
template <> float concatDigitsGeneric(float nb, float newDigit) {
  char buffer[20];
  int len = snprintf(buffer, sizeof(buffer), "%f", nb);
  int lastDigitPos = 0;

  for (auto i = len - 1 ; i > 0 and buffer[i] == '0' ; --i)
    lastDigitPos = i;
  buffer[lastDigitPos] = newDigit + '0';
  buffer[lastDigitPos + 1] = '\0';

  printf("%s: nb = %f, newDigit = %f, buffer = %s\n", __PRETTY_FUNCTION__, nb, newDigit, buffer);
  return atof(buffer);
}
// double (same as float, but we redo the work ...)
template <> double concatDigitsGeneric(double nb, double newDigit) {
  (void)nb; // To suppress the "unused parameter" warning
  (void)newDigit;
  return 0; // Have to duplicate code ... ugly, and error prone
}
template int concatDigitsGeneric<int>(int, int); // Explicit instantiation
template float concatDigitsGeneric<float>(float, float); // Explicit instantiation
template double concatDigitsGeneric<double>(double, double); // Explicit instantiation
```

concatDigits.hh

```
#ifndef CONCAT_DIGITS_HH
# define CONCAT_DIGITS_HH

int concatDigits(int nb, int newDigit);

template <typename T>
T concatDigitsGeneric(T nb, T newDigit);

// template <typename T>
// inline T const& Max (T const& a, T const& b) {
//    return a < b ? b : a;
// }

#endif // !CONCAT_DIGITS_HH
```

Makefile

```
TARGET := concatDigit

SRCS := main_template.cpp concatDigits.cpp
OBJS := $(SRCS:.cpp=.o)

CXXFLAGS += -W -Wall -Wextra -std=c++2a -g3

all: $(TARGET)

$(TARGET): $(OBJS)
    $(CXX) $(CXXFLAGS) $^ -o $@

clean:
    -rm -f $(OBJS)

fclean: clean
    -rm -f $(TARGET)

re: fclean all

test: main_test.o
    $(CXX) $(CXXFLAGS) $^ -o $@
    -$@

.PHONY: all clean fclean re test
```

```
cd "/home/1laurenj/Ensta/Cours ensta IN204/Session 1/cours_3_templates/"
make re
./concatDigit
```

```
## rm -f main_template.o concatDigits.o
## rm -f concatDigit
## g++ -W -Wall -Wextra -std=c++2a -g3   -c -o main_template.o main_template.cpp
## g++ -W -Wall -Wextra -std=c++2a -g3   -c -o concatDigits.o concatDigits.cpp
## g++ -W -Wall -Wextra -std=c++2a -g3 main_template.o concatDigits.o -o concatDigit
## 123 + 5 -> 1235
## 123 + 5 -> 1235
## T concatDigitsGeneric(T, T) [with T = float]: nb = 12.340000, newDigit = 5.000000, buffer = 12.345
## 12.340000 + 5 -> 12.345000
## static float numeric_traits<float>::append(float, T) [with T = float]: aValue = 12.340000, aDigit = 5
## 12.340000 + 5 -> 12.345000
```

Séance 4 : The STL: containers and iterators

Resources

- **Containers** https://isocpp.org/wiki/faq/containers
- **iterators**: https://cplusplus.com/reference/iterator/
- **Full STL documentation**: https://www.cppreference.com/Cpp_STL_ReferenceManual.pdf

Summary of the session:

- Writting a wrapper around an `int` and `char array[];`

```cpp
template <typename T = int /* default */>
class number {
private:
  T c;
public:

  typedef T value_type;

  // main.c: number N1(0);  // 1
  // main.c: number N2(N1); // 2
  // main.c: number N;      // 3
  /* 1 */ number(T const & val) : c(val) {}
  /* 2 */ number(number const & other)
    : c(other.get()) {}
  /* 3 */ number() : c(0) {}

  /* Regular getter/setter */

  T get(/*this,*/ void) const {
    return (c);
  }

  T set(T const & newVal) {
    this->c = newVal;
    return (this->get());
  }

  T set(number const & newVal) {
    this->newVal;
    return (this->get());
  }
};
```

```cpp
/*
** See https://cplusplus.com/reference/array/array/
**
** Template on type T and size S
** size_t is also std::size_t
*/
template <typename T, size_t S>
class myArray {
private:
  T _buffer[S]; // Abstracted array here
public:
  void fill( const T & value ) {
    for (size_t i = 0 ; i < S ; ++i)
      _buffer[i] = value;
  }

  // Regular const getter
  T & get(size_t pos) const {
    return _buffer[pos];
  }

  // operator[] overload (const and non-const)
  // called with: `a[5]`, or `a.operator[](5)`
  T & operator[](size_t pos) {
    return _buffer[pos];
  }

  T const & operator[](size_t pos) const {
    return _buffer[pos];
  }

  size_t size() const { return S; }
};
```

Séance 5 : Operators and streams

Class abstract/wrap features and implement additional functions on it, moreover, unlike **struct** they are considered as a type on its own in the same way as an **int** or **float**

```
template <typename T>
void doStuff(T a, T b) {
  T result = 0;

  a = a + 5;
  a = b * 2;
  result = a + a + a + a + a;
  // etc...
}
```

1. `DoStuff()` is a basic function doing arithmetic

2. We should be able to do that with any type
   * `int`, `short`, `float`, `Class Number`

3. Including more complexe one
   * `char *`, `char &`: Pointers/ref are also a type
   * `char const *(*)(T short, ...)[]` complexeStuff

Main example:

```
#include <iostream>
#include "number.hpp" // Implement this

template <typename T>
void doStuff(T a, T const &pi, int b) {
  T result = b;
  std::string token;
  int tokenCount = 0;

  a.set(a.get() + 5);
  a += pi * 2 + a;
  result = (a += pi) + a + a + a;
  std::cout << result.get() << std::endl; // Getter version (only work with classes)
  std::cout << result << std::endl; // overload version (fully generic)

  for (tokenCount = 0 ; std::operator>>(std::cin, token) ; ++tokenCount); // Complete form
  for (tokenCount = 0 ; operator>>(std::cin, token) ; ++tokenCount); // Simple form
  for (tokenCount = 0 ; std::cin >> token ; ++tokenCount); // Operator form
  std::cout << "found  0" << std::oct << tokenCount << " tokens in the istream (base 8)" << std::endl;
  std::cout << "found   " << std::dec << tokenCount << " tokens in the istream (base 10)" << std::endl;
  std::cout << "found 0x" << std::hex << tokenCount << " tokens in the istream (base 16)" << std::endl;
}

int main() {
  number    nInt = 0;
  number<float> nFloat(1.337);
  number<int> pi = 314;

  doStuff(nInt, pi, (int)nInt);
}
```

Exercice

Upgrade the **class Number** to support operators so it works with the `DoStuff()` function

Complete `class Number` implementation:

```cpp
#include <ostream> // iostream containts too much, we only need the "output" part

template <typename T = int /* default */>
class number {
private:
  T c;
public:
  /* 1 */ number(T const & val) : c(val) {}
  /* 2 */ number(number const & other) : c(other.get()) {}
  /* 3 */ number() : c(0) {}
  /* Operator overload */
  // One-liners (can return T too, because we can construct with it if needed)
  // T      operator+(T const &v) const {return c + v;} // Local scope
  number<T> operator+(number<T> const &o) const {return c + o.c;} // Local scope
  template <typename T1> friend T operator+ (int, number<T1> const & o); // Parent scope

  // This will not compile if we do "a += b * 2 + a;"
  // That's why we need to return this
  // T         operator+(number<T> const &o) const {return c + o.c;}
  number<T> operator*(T const &v) const {return c * v;}

  // Several cases (each returning `*this` to allow a chain of operation):
  // * Regular return
  // * one liner with operator ','
  // * reuse of operator=(T const &)
  number<T> & operator+=(T const &v) { this->c += v; return (*this); }
  number<T> & operator+=(number<T> const &o) { return (c += o.c, *this); }
  number<T> & operator =(T const &v) {return (c = v, *this); }
  number<T> & operator =(number<T> const &o) { return ((*this = o.c), *this); } // Reuse ^

  // Cast operators (adding explicit to denie implicit cast, for the exercice)
  explicit operator int () const { return c; }

  /* Regular getter/setter */
  T get(/*this,*/ void) const {return (c);}
  T set(T const & newVal) {this->c = newVal; return (this->get()); }
  T set(number const & newVal) { this->newVal; return (this->get()); }

  // Over operator<<(): See https://isocpp.org/wiki/faq/input-output#output-operator
  template <typename T1> // Declare external function as friend
  friend std::ostream& operator<< (std::ostream& out, number<T1> const & o);
};

template <typename T> // Implement function
T operator+ (int a, number<T> const & o){return a + o.c;}

template <typename T> // Implement function
std::ostream& operator<< (std::ostream& out, number<T> const & o)
{
  out << "MyValue is: " << o.c; // Can use private directly because of friend
  return out;
}
```

Makefile used

```
TARGET := a.out

SRCS := main.cpp
OBJS := $(SRCS:.cpp=.o)

CXXFLAGS += -W -Wall -Wextra -std=c++17 -g3

all: $(TARGET)

$(TARGET): $(OBJS)
    $(CXX) $(CXXFLAGS) $^ -o $@

clean:
    -rm -f $(OBJS)

fclean: clean
    -rm -f $(TARGET)

re: fclean all

test: main_test.o
    $(CXX) $(CXXFLAGS) $^ -o $@
    -$@

.PHONY: all clean fclean re test
```

```
cd "/home/1laurenj/Ensta/Cours ensta IN204/Session 1/cours_5_operator_stream/"
make re
echo "1 2 3 abcd 5 6 1337 8 9 10 11 12 13 14 15 16 17 18 19 20" | ./a.out
```

```
## rm -f main.o
## rm -f a.out
## g++ -W -Wall -Wextra -std=c++17 -g3   -c -o main.o main.cpp
## g++ -W -Wall -Wextra -std=c++17 -g3 main.o -o a.out
## 3808
## MyValue is: 3808
## found  00 tokens in the istream (base 8)
## found   0 tokens in the istream (base 10)
## found 0x0 tokens in the istream (base 16)
```

Resources

Main Resources:

- **Basic rules and ideoms for operator overload:**
  - https://stackoverflow.com/questions/4421706/what-are-the-basic-rules-and-idioms-for-operator-overloading
- **Operators**:
  - **Operators**: https://en.cppreference.com/w/cpp/language/operator_precedence
  - **Operators**: https://cs.smu.ca/~porter/csc/ref/cpp_operators.html
  - **IO Tutorial**: https://www.learncpp.com/cpp-tutorial/overloading-the-io-operators/
  - **IO Overload**: https://isocpp.org/wiki/faq/operator-overloading
- **IOLibrary**: https://cplusplus.com/reference/iolibrary/
- **IOStream**: https://isocpp.org/wiki/faq/input-output

Séance 6 : C++20: Contracts, specialization and advanced notions

Current course progress reminder:

IN204 : *Programmation Objet & Génie Logiciel*

- ☒ **Séance 1** : Introduction aux objets
- ☒ **Séance 2** : Dérivation & Héritage
- ☒ **Séance 3** : Les Modèles & la Généricité
- ☒ **Séance 4** : The STL: containers and iterators
- ☒ **Séance 5** : Operators and streams**
- ☐ -
- ☐ **Séance 6: C++20: Contracts, specialization and advanced notions** <– We are here
- ☐ -
- ☐ **Séance 7** : Les exceptions
- ☐ **Séance 8** : L'héritage et le polymorphisme
- ☐ **Séance 9** : Parallèlisme & Programmation Asynchrone
- ☐ **Séance 10** : Evaluation au moment de la compilation

Source files

main.cpp

```cpp
#include <iostream> /* std::cout */
#include "defines.hpp" /* For the LOG and LOG_DECL_VAR macro */
#include "prototypes.hpp" /* For the LOG and LOG_DECL_VAR macro */
#include "codelocks.hpp" // Implement this

////////////////////////////////////////
// Toying with concepts

void test_concepts() {
  int i = 1;
  float f = 2.2;
  double d = 4.4;
  custom::Vector v{1,2,3};

  // To remove the "unused variable" warning
  // (we explicitly assess that it is not used, useful when generating code sometime)
  (void)v;

  regular_add(i, i);  // Regular C call
  template_add(i, i); // Deduce template from parameter (int)
  template_add(f, f); //  ``        ``            ``     (float)
  template_add<float>(f, f); // Explicit call of one version

  concept_add_long(i, i); // Also deduce from parameter but using concept
  concept_add_long(f, f); //
  concept_add_long(d, d); //

  concept_add_short(i, i); // Also deduce from parameter but using concept
  concept_add_short(f, f); //
  concept_add_short(d, d); //

  concept_add_short(v, v); //

  // concept_add(v, v); // Concept compiler error
}

////////////////////////////////////////
// Toying with codelocks

namespace cc = ::IN204::codeCrackingExo; // https://en.cppreference.com/w/cpp/language/namespace_alias

template <typename T> requires cc::hasToString<T>
void codelock_counting(T const & codelock)
{
  std::cout << codelock.toString() << std::endl;
}

void test_codelocks() {
  cc::digit d;
  // custom::Vector v;
```

```
  cc::codelock_3_dials three_dials(123);
  cc::codelock_4_dials four_dials(1998);
  cc::digital_5_dials five_dials(31337);

  std::cout << d.toString() << std::endl;
  std::cout << three_dials.toString() << std::endl;
  std::cout << four_dials.toString() << std::endl;
  codelock_counting(five_dials);
  // codelock_counting(v); // Constraints violation, simple error message
}

int main(){
  test_concepts();
  test_codelocks();

  return 0;
}
```

defines.hpp

```
#ifndef DEFINES_HPP_
# define DEFINES_HPP_

// ////////////////////////////////////////////////////////// LOGS + SOME DEFINE
// // C++20 for std::cout formating ala printf (unsupported by compilers yet)
// // Otherwise, see: https://en.cppreference.com/w/cpp/io/manip
// # include <format> /* for std::format() */
# include <iomanip>

// Some colors, because why not
# ifdef USE_COLOR
#  define CLR_RST  "\x1b[0m"
#  define CLR_GRN  "\x1b[32m"
#  define CLR_BLU  "\x1b[34m"
#  define CLR_YEL  "\x1b[33m"
#  define CLR_BOLD "\x1b[1m"
# else
#  define CLR_RST  ""
#  define CLR_GRN  ""
#  define CLR_BLU  ""
#  define CLR_YEL  ""
#  define CLR_BOLD ""
# endif // !USE_COLOR

// In case the pretty_function macro is not defined (ex: visual studio on windows)
# if !defined(__PRETTY_FUNCTION__) && !defined(__GNUC__)
#  define __PRETTY_FUNCTION__ __FUNCSIG__
# endif

// Some inline logging to simplify the code later during debug
# define LOG_DECL_VAR static size_t g_log_line; // Zeroed by default because of the static keyword
# define LOG(v) (std::cout << "[" << std::setw(2) << ++g_log_line << "] " \
        << CLR_BLU CLR_BOLD << __FILE__ << CLR_RST     \
        << ":"   << CLR_YEL << __LINE__  << CLR_RST        \
        << ":\t" << CLR_GRN <<  __PRETTY_FUNCTION__ << CLR_RST \
        << "{" << #v << " = " << (v) << "}"             \
        << std::endl)

LOG_DECL_VAR; // To instanciate the static global variable (for the log line number)

# define ADD_CODE { LOG(a + b); return a + b; }

#endif /* !DEFINES_HPP_ */
```

prototypes.hpp

```hpp
#ifndef PROTOTYPES_HPP_
# define PROTOTYPES_HPP_

# include <ostream> // std::ostream

namespace custom { // A toy namespace
  struct Vector {
    int x;
    int y;
    int z;
    Vector operator+(auto const &o) const {
      return Vector{x + o.x, y + o.y, z + o.z};
    }

    // // Compilation error without the cast operator:
    // defines.hpp:33:42: error: cannot convert ‘custom::Vector’ to ‘int’ in return
    //     33 | # define ADD_CODE { LOG(a + b); return a + b; }
    operator int () const { return this->x; } // for "return (Vector + Vector);" to works

    friend std::ostream& operator<< (std::ostream& out, Vector const & o) {
      return out << "\n\t{x=" << o.x << ", y=" << o.y << ", z=" << o.z << "}";
    }
  };

  // Creating some concept as a general exercice
  template <typename T> concept addable = requires(T a, T b){{a + b};};
  template <typename T> concept isNotIntOrFloat = !(std::integral<T> || std::floating_point<T>);

} // !namespace custom

/*   ******************   */ int  regular_add(int a, int b) ADD_CODE
template <typename T>         int template_add(T   a, T   b) ADD_CODE
template <> /* Specialized */ int template_add(float a, float b) ADD_CODE

// Full syntax
template <typename T> requires std::integral<T>           int concept_add_long(T a, T b) ADD_CODE
template <typename T> requires std::floating_point<T>     int concept_add_long(T a, T b) ADD_CODE
template <typename T> requires custom::isNotIntOrFloat<T> int concept_add_long(T a, T b) ADD_CODE

// Abreviation syntax (we can use typename, class, or now a constraint for T)
template <std::integral T> /***/     int concept_add_short(T a, T b) ADD_CODE
template <std::floating_point T>     int concept_add_short(T a, T b) ADD_CODE
template <custom::isNotIntOrFloat T> int concept_add_short(T a, T b) ADD_CODE

#endif /* ! PROTOTYPES */
```

codelocks.hpp

```cpp
#ifndef CODELOCKS_HPP_
# define CODELOCKS_HPP_

# include <string>
# include <array>

namespace IN204 {
  namespace codeCrackingExo {

    template <typename T>
    concept hasToString = requires (T t)
      {
       t.toString();
      };

    ////////////////////////////////////////////////
    /// digit

    class digit {
    private:
      char d;
      std::string const base;
    public:
      digit(char d = 0, std::string const & base = "0123456789") : d(d), base(base) {}

      std::string toString() const { return std::string() + base[d % base.length()]; }
      digit &operator=(int v) { return (d = v, *this); }
    }; // !class

    ////////////////////////////////////////////////
    /// codelock

    template <std::size_t S, typename D = digit>
    class codelock {
    private:
      std::array<D, S> code;
    public:
      // @param defaultCode The value to initialize the codelock at
      codelock(int defaultCode) : code{0} {
    // Initialize
    for (auto it = code.rbegin(); (it != code.rend() && defaultCode != 0); ++it) {
      *it = (defaultCode % 10);
      defaultCode /= 10;
    }
    // if (defaultCode != 0)
    //   std::throw // TODO
      }

      std::string toString() const {
    std::string rval;

    for (auto const & item : code)
```

```
      rval += item.toString();
    return rval;
      }
    }; // !class

    ////////////////////////////////////////////////
    /// aliases

    template <std::size_t S, typename D = digit>
    class digitalCodelock : public codelock<S, D> {
    };

    template <std::size_t S, typename D = digit>
    class verboseCodelock : public codelock<S, D> {
    };

    using codelock_3_dials = codelock<3>;
    using codelock_4_dials = codelock<4>;
    using digital_5_dials = digitalCodelock<5>;

  } // ! codeCracking
} // !IN204

// typedef 4dials_combo codelock<>;

#endif /* ! CODELOCKS_HPP_ */
```

Makefile used

```
TARGET := a.out
SRCS := main.cpp
OBJS := $(SRCS:.cpp=.o)

# Ubuntu and MinGW: sudo apt-get install gcc-10 g++-10
CXX = g++-10 # overwrite default g++ on my system which is version 9
CXXFLAGS += -W -Wall -Wextra -std=c++20

all: $(TARGET)

color: CXXFLAGS += -DUSE_COLOR
color: fclean all


$(TARGET): $(OBJS)
    $(CXX) $(CXXFLAGS) $^ -o $@

clean:
    -rm -f $(OBJS)

fclean: clean
    -rm -f $(TARGET)

re: fclean all

.PHONY: all clean fclean re test color
```

```
cd "/home/1laurenj/Ensta/Cours ensta IN204/Session 1/cours_6_Cpp20_and_contracts/"
make re
./a.out
```

```
## rm -f main.o
## rm -f a.out
## g++-10  -W -Wall -Wextra -std=c++20   -c -o main.o main.cpp
## g++-10  -W -Wall -Wextra -std=c++20 main.o -o a.out
## [ 1] prototypes.hpp:31:  int regular_add(int, int){a + b = 2}
## [ 2] prototypes.hpp:32:  int template_add(T, T) [with T = int]{a + b = 2}
## [ 3] prototypes.hpp:33:  int template_add(T, T) [with T = float]{a + b = 4.4}
## [ 4] prototypes.hpp:33:  int template_add(T, T) [with T = float]{a + b = 4.4}
## [ 5] prototypes.hpp:36:  int concept_add_long(T, T) [with T = int]{a + b = 2}
## [ 6] prototypes.hpp:37:  int concept_add_long(T, T) [with T = float]{a + b = 4.4}
## [ 7] prototypes.hpp:37:  int concept_add_long(T, T) [with T = double]{a + b = 8.8}
## [ 8] prototypes.hpp:41:  int concept_add_short(T, T) [with T = int]{a + b = 2}
## [ 9] prototypes.hpp:42:  int concept_add_short(T, T) [with T = float]{a + b = 4.4}
## [10] prototypes.hpp:42:  int concept_add_short(T, T) [with T = double]{a + b = 8.8}
## [11] prototypes.hpp:43:  int concept_add_short(T, T) [with T = custom::Vector]{a + b =
##  {x=2, y=4, z=6}}
## 0
## 123
## 1998
## 31337
```

Resources

- **Official course:**: https://perso.ensta-paris.fr/~bmonsuez/Cours/doku.php?id=in204:seances:seance6
- **C++2a and constraints**:
  - **isocpp guide**: https://isocpp.org/blog/2021/11/cpp-20-concepts (very good)
  - **cppreference**: https://en.cppreference.com/w/cpp/language/constraints
  - **others**: https://www.cppstories.com/2021/concepts-intro/
- **Iterators**:
  - **Link 1**: https://www.geeksforgeeks.org/introduction-iterators-c/
  - **Link 2**: https://www.geeksforgeeks.org/iterators-c-stl/
  - **Custom iterators**: https://www.internalpointers.com/post/writing-custom-iterators-modern-cpp

Course 7 : Error management and exceptions

Error management

Source files

```
#ifndef SCREEN_HPP_
# define SCREEN_HPP_

# include <vector>
# include "Pixel.hpp"

// To test `Screen(someRandomStruct iAmObviouslyNotASize)`
// Note: "new Pixel[iAmObviouslyNotASize]" works because
//        we got a cast operator here (implicit convertion)
// Note2: Don't do that in your project :p, it's for the test
struct someRandomStruct {
  operator int() { return -1; }
};

class Screen {
  // By default, every attribute is private within a class
  // (but we explicit the 'private:' anyway for readability
private:
  // Using a vector would be better, but we do it also by hand for
  // the exercice with new/delete.
  std::vector<Pixel> pixels_vector;

  Pixel * pixels_manual;
  size_t size;
public:
  // https://en.cppreference.com/w/cpp/language/nullptr
  // https://en.cppreference.com/w/cpp/language/new
  Screen(size_t size = 0) : pixels_manual(nullptr), size(size) {
    pixels_manual = new Pixel[size]; // Can throw
    pixels_vector.resize(size);
  }
  Screen(someRandomStruct iAmObviouslyNotASize) {
    std::cout << __PRETTY_FUNCTION__ << ": Before exception" << std::endl;
    pixels_manual = new Pixel[iAmObviouslyNotASize]; // Will throw std::bad_array_new_length
    std::cout << __PRETTY_FUNCTION__ << ": After exception" << std::endl;
  }
  // NO DEFAULT, OR WE WILL GET MEMORY CORRUPTION ON THE SECOND DESTRUCTOR
  // --> { Screen a(10); Screen b(a); } // Program could crash here (double memory free)
  // Screen(Pixel const &) = default;
  ~Screen() {
    delete pixels_manual;
    // No need to delete pixels_vector, it will get destroyed implicitly
    // (because Vector<> has a destructor that will get called)
  }
  auto operator=(Screen const &o) -> Screen & {
     // Vector version
     { // Just an extra local stack scope (to group code and prevent local variable to spread)
      pixels_vector = o.pixels_vector;
     }
     { // Just an extra local stack scope (--)
```

```
      // Manual version
      auto tmp = new Pixel[size]; // Can throw
      // We delete AFTER (in case the new operator throws, so the class Screen never has invalid memory
      delete pixels_manual;
      // Old C style copy (but no memcpy or std::copy, we don't know if Pixel is a PoD)
      // See https://en.cppreference.com/w/cpp/language/classes#POD_class
      for (size = 0 ; size < o.size ; ++size)
    tmp[size] = o.pixels_manual[size]; // pixel.operator=(...) noexcept
      pixels_manual = tmp; // Assign once everything is ready and return safely
     }
     return *this;
    }
}; // !class

#endif /* !SCREEN_HPP_ */
```

Screen.hpp

```
#ifndef PIXEL_HPP_
# define PIXEL_HPP_

class Pixel {
private:
  std::uint32_t v;
  int rgb[3]; // To show how to init an array by hand in the constructor (below)

public:

  // The 4 methods to respect the Coplian form (Default CTor/DTor + copy + operator=()
  // (CTors/DTors == ConsTructor/DesTructor)
  Pixel(std::uint32_t argb = 0) : v(argb), // regular constructor
                  rgb{0,1,2} // Initializer list needed because multiple items
  {
    std::cout << "constructing Pixel with R:" << rgb[0]
          << " / G:" << rgb[1]
          << " / B:" << rgb[2] << std::endl;

  }
  Pixel(Pixel const &) = default;
  ~Pixel() = default; // Defaulted, a memory copy works here
  auto operator=(Pixel const &) noexcept -> Pixel & = default; // Alternate auto syntax

  //////////

  auto A() const -> std::uint8_t { return ((v >> 8*0) & 0xFF); }
  auto R() const -> std::uint8_t { return ((v >> 8*1) & 0xFF); }
  auto G() const -> std::uint8_t { return ((v >> 8*2) & 0xFF); }
  auto B() const -> std::uint8_t { return ((v >> 8*3) & 0xFF); }

}; //!class

#endif /* !PIXEL_HPP_ */
```

Pixel.hpp

```cpp
#include <iostream>
#include "Screen.hpp"
#include "Pixel.hpp"


// using namespace std; // Don't do that, please ... just don't
// See: https://isocpp.github.io/CppCoreGuidelines/CppCoreGuidelines#Rs-using

// @return the value of Pi
float i_am_bad_at_math() {
  float yes = 1;
  float ofcourse = 2;
  return (1 + 1 == 3 ? yes : ofcourse); // Returns the value of Pi
}

// @return 0 on succcess, -1 on failure
int classic_error_management() {
  float homework;
  float isGood;

  isGood = 3.14; // Reference value, sometime it's easier to read (avoid hardcoded unnamed values)
  homework = i_am_bad_at_math();
  // if (homework == 3.14) { // Avoid hardcoded values as much as possible, name them !
  if (homework == isGood) {
    return 0;
  }
  return -1; // Any case that is not explictly a success is a failure, by default (safer)
}

// @brief A test function to toy with exceptions
void testException__bad_array_new_length()
{
  try { // 1
    try { // 2
      someRandomStruct testStruct;
      Screen s(testStruct); // Exception here
    } catch (std::exception& e) { // 2
      std::cout << __PRETTY_FUNCTION__ << ": Catching exception n'1 and rethrowing" << std::endl;
      throw; // re-throw the current exception for fun
    }
  } catch (std::exception& e) { // 1
    std::cout << __PRETTY_FUNCTION__ << ": Catching exception n'2 and ignoring" << std::endl;
    throw; // re-catching, and rethrow for the caller to handle this
  }
}

int main(){


  std::cout << "==================" << std::endl;
  std::cout << "== main() called" << std::endl << std::endl;
```

```
  Pixel a; // To test the Pixel constructor cout

  try {
    testException__bad_array_new_length();
  } catch (...) { // A catch all guard
    std::cout << __PRETTY_FUNCTION__ << ": "
          << "Called code is rethrowing to the caller function as expected"
          << std::endl;
  }

  ///// Classic error management from caller function

  {
    int rval = classic_error_management();
    std::cout << __PRETTY_FUNCTION__ << ": classic_error_management returned a " << (rval == -1 ? "Failu
  }

  std::cout << std::endl << "==================" << std::endl;
  return 0;
}
```

main.cpp

```
TARGET := a.out
SRCS := main.cpp
OBJS := $(SRCS:.cpp=.o)

# Ubuntu and MinGW: sudo apt-get install gcc-10 g++-10
CXX = g++-10 # overwrite default g++ on my system which is version 9
CXXFLAGS += -W -Wall -Wextra -std=c++20

all: $(TARGET)

color: CXXFLAGS += -DUSE_COLOR
color: fclean all

$(TARGET): $(OBJS)
    $(CXX) $(CXXFLAGS) $^ -o $@

clean:
    -rm -f $(OBJS)

fclean: clean
    -rm -f $(TARGET)

re: fclean all

.PHONY: all clean fclean re test color
```

Makefile used

```
cd "/home/1laurenj/Ensta/Cours ensta IN204/Session 1/cours_7_exceptions/"
make re
valgrind ./a.out ## Calling with valgrind to check for memory leaks
## Pay attention to that line: "All heap blocks were freed -- no leaks are possible"
```

Running output

```
## rm -f main.o
## rm -f a.out
## g++-10  -W -Wall -Wextra -std=c++20   -c -o main.o main.cpp
## g++-10  -W -Wall -Wextra -std=c++20 main.o -o a.out
## ==71183== Memcheck, a memory error detector
## ==71183== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
## ==71183== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
## ==71183== Command: ./a.out
## ==71183==
## ==================
## == main() called
##
## constructing Pixel with R:0 / G:1 / B:2
## Screen::Screen(someRandomStruct): Before exception
## void testException__bad_array_new_length(): Catching exception n'1 and rethrowing
## void testException__bad_array_new_length(): Catching exception n'2 and ignoring
## int main(): Called code is rethrowing to the caller function as expected
## int main(): classic_error_management returned a Failure
## ==================
## ==71183==
## ==71183== HEAP SUMMARY:
## ==71183==     in use at exit: 0 bytes in 0 blocks
## ==71183==   total heap usage: 3 allocs, 3 frees, 76,936 bytes allocated
## ==71183==
## ==71183== All heap blocks were freed -- no leaks are possible
## ==71183==
## ==71183== For lists of detected and suppressed errors, rerun with: -s
## ==71183== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Resources

- Entry slides:
  - **Exception safety**: https://en.wikipedia.org/wiki/Exception_safety
  - **Error Handling, by David Svoboda**: https://resources.sei.cmu.edu/asset_files/Presentation/2016_017_101_484207.pdf
  - **Exception safety concept**: https://www.stroustrup.com/except.pdf
- Main documentation:
  - **Exceptions (guide isocpp)**: https://isocpp.org/wiki/faq/exceptions
  - **Exceptions (google coding style)**: https://google.github.io/styleguide/cppguide.html#Exceptions
  - **C++ Coding guidelines**: https://isocpp.github.io/CppCoreGuidelines/CppCoreGuidelines
  - **Reference documentation with exercises (hard)**:https://stroustrup.com/
  - **Boost exception review**: https://www.boost.org/community/exception_safety.html
  - **Official course:**: https://perso.ensta-paris.fr/~bmonsuez/Cours/doku.php?id=in204:seances:seance7
- Remember when I talked about the different stages of a variable (allocation, initialization, usage and cleanup) along with good programming practices ? here are some good readings:
  - **RAII**: https://en.cppreference.com/w/cpp/language/raii
  - **Rule of 3**: https://en.cppreference.com/w/cpp/language/rule_of_three
  - **man errno**: https://man7.org/linux/man-pages/man3/errno.3.html
- https://github.com/JBL-Repo/IN204/blob/main/cours_recap.pdf

Course 8 : Parallel programming

Source files

main.cpp (parellel test)

```cpp
#include <iostream>
#include <sstream>
#include <string>

#include <chrono>

#include <thread>
#include <vector>
#include <mutex>
#include <condition_variable>

std::mutex g_mutex;

void sleep_ms(size_t ms) {
  std::this_thread::sleep_for(std::chrono::milliseconds(ms));
}

void wait_for_order(int i, bool const & do_action, void (*action)(int)) {
  std::stringstream logMsg;

  std::cout << "   COUT: Thread " << i << " -" // Multiple operator<<() called
        << " is running with id " << std::this_thread::get_id()  << std::endl;
  logMsg    << "SSTREAM: Thread " << i << " -" // operator<<() called once
        << " is running with id " << std::this_thread::get_id() << std::endl;
  // "Atomic" print
  std::cout << logMsg.str();

  // "spinLock"
  while (!do_action) {
  }

  // All at once
  action(i);

  sleep_ms(500);

  // One at a time
  { // stack scope for the mutex guard
    std::lock_guard<std::mutex> lock(g_mutex);
    action(i*-1);
  }

}

int main() {
  int numThreads = 5;
  bool do_action = false;
  std::vector<std::thread> threads;
```

```
  std::cout << "[ ] Main thread started (parrallel testing)" << std::endl;

  for (int i = 0 ; i < numThreads ; ++i) {
    // https://en.cppreference.com/w/cpp/thread/thread/thread
    // https://isocpp.org/wiki/faq/cpp11-language#lambda
    // https://en.cppreference.com/w/cpp/utility/functional/ref
    threads.push_back(std::thread(wait_for_order, i, std::cref(do_action),
                  [](int i){std::cout << "Hello from " << i << std::endl;; }));
  }

  sleep_ms(100); // Just to be sure all threads have printed, for the demo
  std::cout << "Main thread with id " << std::this_thread::get_id() << " is sleeping" << std::endl;
  sleep_ms(1000);

  std::cout << "[ ] Main thread is releasing the threads: (spammy action)" << std::endl << std::endl;
  do_action = true; // Trigger the threads
  sleep_ms(250);
  std::cout << std::endl << "[ ] Async call of action should be done by now (clean)" << std::endl << st

  while (threads.size()) {
    threads.back().join(); // main process thread waits for the thread to finish
    threads.pop_back();
  }

  std::cout << "[ ] Main thread is now returning (exit)" << std::endl << std::endl;
  return 0;
}
```

diamond.cpp (diamond test)

```cpp
#include <iostream>

// https://stackoverflow.com/questions/23360572/c-diamond-inheritance-constructor

class compteur {
protected:
  int i;
public:
  compteur(int i) : i(i) {}
  // Virtual will point the the last class implementation (unless final is specified)
  virtual void hello() const {std::cout << i << " - Hello from " << __PRETTY_FUNCTION__ << std::endl; }
};

// // "virtual public compteur" is needed to prevent the diamond issue
// 1laurenj@frpfaluxu003:~/Ensta/Cours ensta IN204/Session 1/cours_8_parallel$ g++ -W -Wall -Wextra dia
// diamond.cpp: In function ‘int main()’:
// diamond.cpp:29:18: error: ‘compteur’ is an ambiguous base of ‘compteur_bi’
//    29 |   compteur &c4 = c_bi;

struct compteur_inc : virtual public compteur {
  compteur_inc() : compteur(1) {}
  void hello() const {std::cout << i << " - Hello from " << __PRETTY_FUNCTION__ << std::endl; }
};

struct compteur_dec : virtual public compteur {
  compteur_dec() : compteur(-1) {}
  void hello() const {std::cout << i << " - Hello from " << __PRETTY_FUNCTION__ << std::endl; }
};

struct compteur_bi : public compteur_inc, public compteur_dec { // inc then dec (for the test)
  // Because of virtual public, base class HAS to be constructed here (because otherwise it would be cr
  compteur_bi() : compteur(2) {}
  void hello() const {std::cout << compteur_inc::i << " - Hello from " << __PRETTY_FUNCTION__ << std::en
};
struct compteur_bi2 : public compteur_dec, public compteur_inc { // dec then inc (for the test)
  // Because of virtual public, base class HAS to be constructed here (because otherwise it would be cr
  compteur_bi2() : compteur(-2) {}
  void hello() const {std::cout << compteur_dec::i << " - Hello from " << __PRETTY_FUNCTION__ << std::en
};

int main() {

  std::cout << "=== Diamond shape testing "<< std::endl;

  compteur c(0);
  compteur_inc c_inc;
  compteur_dec c_dec;
  compteur_bi c_bi;
  compteur_bi2 c_bi2;

  compteur &c1 = c;
  compteur &c2 = c_inc;
```

```
  compteur &c3 = c_dec;
  compteur &c4 = c_bi;
  compteur &c5 = c_bi2;

  std::cout << "==="<< std::endl;

  c.hello();
  c_inc.hello();
  c_dec.hello();
  c_bi.hello();
  c_bi2.hello();

  std::cout << "==="<< std::endl;

  c1.hello();
  c2.hello();
  c3.hello();
  c4.hello();
  c5.hello();

  std::cout << "==="<< std::endl;

  return 0;
}
```

Makefile used

```
TARGET_PARALLEL := parallel
TARGET_DIAMOND := diamond

SRCS_PARALLEL := main.cpp
OBJS_PARALLEL := $(SRCS_PARALLEL:.cpp=.o)

SRCS_DIAMOND := diamond.cpp
OBJS_DIAMOND := $(SRCS_DIAMOND:.cpp=.o)

# Ubuntu and MinGW: sudo apt-get install gcc-10 g++-10
CXX = g++-10 # overwrite default g++ on my system which is version 9
CXXFLAGS += -W -Wall -Wextra -std=c++20 -pthread

all: $(TARGET_PARALLEL) $(TARGET_DIAMOND)

$(TARGET_DIAMOND): $(OBJS_DIAMOND)
    $(CXX) $(CXXFLAGS) $^ -o $@
$(TARGET_PARALLEL): $(OBJS_PARALLEL)
    $(CXX) $(CXXFLAGS) $^ -o $@

clean:
    -rm -f $(OBJS_PARALLEL) $(OBJS_DIAMOND)

fclean: clean
    -rm -f $(TARGET_PARALLEL) $(TARGET_DIAMOND)

re: fclean all

.PHONY: all clean fclean re test color
```

Running output

```
cd "/home/1laurenj/Ensta/Cours ensta IN204/Session 1/cours_8_parallel/"
make re
./parallel
./diamond
```

```
## rm -f main.o diamond.o
## rm -f parallel diamond
## g++-10  -W -Wall -Wextra -std=c++20 -pthread   -c -o main.o main.cpp
## g++-10  -W -Wall -Wextra -std=c++20 -pthread main.o -o parallel
## g++-10  -W -Wall -Wextra -std=c++20 -pthread   -c -o diamond.o diamond.cpp
## g++-10  -W -Wall -Wextra -std=c++20 -pthread diamond.o -o diamond
## [ ] Main thread started (parrallel testing)
##    COUT: Thread    COUT: Thread 01 - is running with id  - is running with id 1406389251253761406389
##
##    COUT: Thread 3 - is running with id 140638899947264
## SSTREAM: Thread 0 - is running with id 140638925125376
## SSTREAM: Thread 1 - is running with id 140638916732672
## SSTREAM: Thread 3 - is running with id 140638899947264
##    COUT: Thread 4 - is running with id 140638891554560
## SSTREAM: Thread 4 - is running with id 140638891554560
##    COUT: Thread 2 - is running with id 140638908339968
## SSTREAM: Thread 2 - is running with id 140638908339968
## Main thread with id 140638925129536 is sleeping
## [ ] Main thread is releasing the threads: (spammy action)
##
## Hello from Hello from Hello from Hello from 4
## 1Hello from
## 3
## 0
## 2
##
## [ ] Async call of action should be done by now (clean)
##
## Hello from 0
## Hello from -4
## Hello from -1
## Hello from -3
## Hello from -2
## [ ] Main thread is now returning (exit)
##
## === Diamond shape testing
## ===
## 0 - Hello from virtual void compteur::hello() const
## 1 - Hello from virtual void compteur_inc::hello() const
## -1 - Hello from virtual void compteur_dec::hello() const
## 2 - Hello from virtual void compteur_bi::hello() const
## -2 - Hello from virtual void compteur_bi2::hello() const
## ===
## 0 - Hello from virtual void compteur::hello() const
## 1 - Hello from virtual void compteur_inc::hello() const
## -1 - Hello from virtual void compteur_dec::hello() const
## 2 - Hello from virtual void compteur_bi::hello() const
```

```
## -2 - Hello from virtual void compteur_bi2::hello() const
## ===
```

Resources

- **Threads:** https://en.wikipedia.org/wiki/Thread_(computing)
- **Virtuals:** https://isocpp.org/wiki/faq/virtual-functions
- **Concurrency:** https://isocpp.org/wiki/faq/cpp11-library-concurrency
- **Pipeline (optional)**: https://en.wikipedia.org/wiki/Instruction_pipelining

Course 10 : Project good practices

Resources

- Project structure
  - **Basics**: https://www.open-std.org/jtc1/sc22/wg21/docs/papers/2018/p1204r0.html
  - **Pitchfork**: https://api.csswg.org/bikeshed/?force=1&url=https://raw.githubusercontent.com/vector-of-bool/pitchfork/develop/data/spec.bs
- CMake (Cross-platform, compiler-independent build system generator):
  - **Begginer Guide**: https://cmake.org/cmake/help/latest/guide/tutorial/index.html
  - **Official guide**: https://cmake.org/cmake/help/latest/guide/tutorial/A%20Basic%20Starting%20Point.html
- Documentation:
  - **Doxygen**: https://doxygen.nl/manual/docblocks.html
  - **Doxygen guide**: https://franckh.developpez.com/tutoriels/outils/doxygen/
- Examples of coding standards:
  - **Google**: https://google.github.io/styleguide/cppguide.html
  - **BDE**: https://bloomberg.github.io/bde/knowledge_base/coding_standards.html
  - **LLVM**: https://llvm.org/docs/CodingStandards.html
- Extra reading:
  - **C++ guidelines**: https://isocpp.org/wiki/faq/coding-standards
  - **Const FAQ**: https://isocpp.org/wiki/faq/const-correctness
  - **Cppcon pdf on "const"**: https://raw.githubusercontent.com/CppCon/CppCon2015/master/Presentations/The%20Importance%20of%20Being%20Const/The%20Importance%20of%20Being%20Const%20-%20Richard%20Powell%20-%20CppCon%202015.pdf

Exam 2020 correction

- **Subject**: ttps://perso.ensta-paris.fr/~bmonsuez/Cours/lib/exe/fetch.php?media=in204:8_-*in204*-_examen.pdf

Source file

```cpp
#include <iostream>
#include <string>
#include <complex>
#include <vector>
#include <cstdio>
#include <algorithm>

// class given declaration
class key_value_pair {
private:
  int key;
  std::string value;
public:
  key_value_pair();
  key_value_pair(int theKey, std::string theValue);
  key_value_pair(const key_value_pair& anotherPair);
  // Question 1.1: https://cplusplus.com/doc/tutorial/classes2/
  // key_value_pair(const key_value_pair& anotherPair) = default; // Implicitly declared

  // Question 2:
  // For more info on constness:
  // * https://isocpp.org/wiki/faq/const-correctness
  // * https://raw.githubusercontent.com/CppCon/CppCon2015/master/Presentations/The%20Importance%20of%2
  int getKey() const { return key; }
  int setKey(int k) { key = k; return key; } // we return key for conveniance
  // int getKey(/*this*/) const { return this->key; } // Would work too

  std::string getValue() const { return value; }
  std::string setValue(std::string const &v) { value = v; return value; }

  // Question 3:
  // No need to name the parameter when it is only a prototype
  bool operator == (const key_value_pair&) const;
  bool operator != (const key_value_pair&) const;

  bool operator < (const key_value_pair&) const;
  bool operator > (const key_value_pair&) const;
  bool operator <=(const key_value_pair&) const;
  bool operator >=(const key_value_pair&) const;
};

// Question 4: templates
template <typename keyT, typename valueT>
class templated_kv_class {
private:
  keyT key;
  valueT value;
public:
  // Just redoing the getters to illustrate
  templated_kv_class() : key(0), value("1337") {};
  templated_kv_class(keyT const &k, valueT const &v) : key(k), value(v) {};
  keyT getKey() const { return key; }
```

```
  keyT setKey(keyT k) { key = k; return key; }
  valueT getValue() const { return value; }
  valueT setValue(valueT const &v) { value = v; return value; }
  // Question 5:
  // sort by value and not key, for the exercice
  bool operator < (const templated_kv_class<keyT,valueT> & o) const {return (value < o.value);}

  // Question 6:
  template<class charT, class traits>
  friend std::basic_ostream<charT, traits>& operator << (std::basic_ostream<charT, traits>& aStream,
                              templated_kv_class<keyT, valueT> const &thePair) {
  // Question 6.2: Custom print
    if (thePair.getKey() == "du" && thePair.getValue() == 40)
      aStream << "du => 40";
    else
      aStream << "{key:" << thePair.getKey() << ", value: " << thePair.getValue() << "}";
    return aStream; // So we can use several "<<" in a row, for conveniance
  }
};

template <typename K, typename V>
class key_defined_value : public templated_kv_class<K, V> {
public:
  bool is_void;
  key_defined_value() : templated_kv_class<K, V>(), is_void(true) {};
  key_defined_value(K const &k, V const &v) : templated_kv_class<K, V>(k, v), is_void(false) {};

  // Call parent setKey() method and set our is_void then
  K setKey(K k) {is_void = false; return templated_kv_class<K, V>::setKey(k); }
};

// Definitions
key_value_pair::key_value_pair() : key(0), value() {}
key_value_pair::key_value_pair(int k, std::string v) : key(k), value(v) {}
key_value_pair::key_value_pair(const key_value_pair&o) : key(o.key), value(o.value) {}

// Question 2:
// It is not possible to access private attributes without "friend" or getters.

// Question 3: Only operator=() and operator<() need to be "technically" implemented
bool key_value_pair::operator == (const key_value_pair&o) const {
  return (key == o.key && value == o.value);
}
// Subject doesn't tell how to sort, so let's only sort by key here
bool key_value_pair::operator < (const key_value_pair&o) const {
  return (key < o.key);
}
bool key_value_pair::operator !=(const key_value_pair&o) const { return !(*this == o); }
bool key_value_pair::operator <=(const key_value_pair&o) const { return (*this == o || *this < o); }
bool key_value_pair::operator > (const key_value_pair&o) const { return (*this != o) && !(*this < o); }
bool key_value_pair::operator >=(const key_value_pair&o) const { return (*this == o  || !(*this < o)); }

int main() {
```

```
key_value_pair kv1;
key_value_pair kv2;

std::cout << "== Testing the 2021 exam correction ==" << std::endl;
std::cout << std::endl;
std::cout << "Q2: kv1 default values ==> "
      << "{key:" << kv1.getKey()
      << ", value: \"" << kv1.getValue() <<  "\"} "
      << std::endl;

kv1.setValue("This is SPARTAAAA");
kv2.setValue("No, this is Patrick !"); // because why not

std::cout << "Q2: kv1 new value ==> "
      << "{key:" << kv1.getKey()
      << ", value: \"" << kv1.getValue() <<  "\"} "
      << std::endl;

std::cout << std::endl;
std::cout << "Q3: Are kv1 and kv2 equal ? " << std::boolalpha << (kv1 == kv2) << std::endl;
std::cout << "Q3: Are kv1 and kv2 different ? " << std::boolalpha << (kv1 != kv2) << std::endl;

kv1.setKey(1);
kv2.setKey(2);

std::cout << "Q3: kv1 new values ==> "
      << "{key:" << kv1.getKey()
      << ", value: \"" << kv1.getValue() <<  "\"} "
      << std::endl;
std::cout << "Q3: kv2 new values ==> "
      << "{key:" << kv2.getKey()
      << ", value: \"" << kv2.getValue() <<  "\"} "
      << std::endl;

std::cout << "Q3: kv1 == kv2: " << std::boolalpha << (kv1 == kv2) << std::endl;
std::cout << "Q3: kv1 != kv2: " << std::boolalpha << (kv1 != kv2) << std::endl;
std::cout << "Q3: kv1 >  kv2: " << std::boolalpha << (kv1 >  kv2) << std::endl;
std::cout << "Q3: kv1 <  kv2: " << std::boolalpha << (kv1 <  kv2) << std::endl;
std::cout << "Q3: kv1 >= kv2: " << std::boolalpha << (kv1 >= kv2) << std::endl;
std::cout << "Q3: kv1 <= kv2: " << std::boolalpha << (kv1 <= kv2) << std::endl;
std::cout << std::endl;

// Question 4: templates
templated_kv_class<int, std::string> kv_templated;

std::cout << "Q4: kv_templated new values ==> "
      << "{key:" << kv_templated.getKey()
      << ", value: \"" << kv_templated.getValue() <<  "\"} "
      << std::endl;

std::cout << std::endl;

// Question 4.2: Doesn't compile because complex is template itself
```

```
  // * see https://en.cppreference.com/w/cpp/numeric/complex (it is a litteral type)
  // templated_kv_class<std::complex, std::string> kv_templated2; // KO
  templated_kv_class<std::complex<float>, std::string> kv_templated2; // OK
  // Comparison operators are not implemented, seems like a math problem ? (unsure about that one)

  // Question 5
  // The following code instanciate a class and adds it to the end of the std::vector<>
  // * see https://en.cppreference.com/w/cpp/container/vector/push_back
  std::vector<templated_kv_class<std::string, float>> listOfIdentifiers;
  listOfIdentifiers.push_back(templated_kv_class<std::string, float>("mot", 10));
  listOfIdentifiers.push_back(templated_kv_class<std::string, float>("le", 100));
  listOfIdentifiers.push_back(templated_kv_class<std::string, float>("la", 80));
  listOfIdentifiers.push_back(templated_kv_class<std::string, float>("du", 40));

  // With printf for a change
  int i = 0;
  for (auto it = listOfIdentifiers.cbegin() ; it != listOfIdentifiers.cend() ; ++it)
    printf("Q5: [%d/%zu] key: %s, value: %f\n", ++i, listOfIdentifiers.size(), it->getKey().c_str(), it

  // Question 5.2: Sorting
  // We can sort the vector by many mean, but the subject implies using a function for that.
  // * See https://cplusplus.com/reference/algorithm/sort/
  std::cout << std::endl;
  std::cout << "Q5: Sorting out the vector by key with the operator<() and std::sort()" << std::endl;

  int j = 0;
  std::sort(listOfIdentifiers.begin(), listOfIdentifiers.end());
  for (auto it = listOfIdentifiers.cbegin() ; it != listOfIdentifiers.cend() ; ++it)
    printf("Q5: [%d/%zu] key: %s, value: %f (sorted by value)\n", ++j, listOfIdentifiers.size(), it->ge

  for (auto it = listOfIdentifiers.cbegin() ; it != listOfIdentifiers.cend() ; ++it)
    std::cout << "Q5: Using operator<<() this time: " << *it << std::endl;


  // Question 6: Stream Operator<<()
  // * This operator can't be defined in the class because it is a global binary operator.
  //   Otherwise it would be the class itself which would print, it needs to be the other way around.
  //   (don't forget that it is someone else who is printing us).

  // Question 7:
  key_defined_value<int, std::string> kdv1;

  std::cout << std::endl;

  std::cout << "Q7: Is kdv1 value unset (void ?) ? " << std::boolalpha << kdv1.is_void << ". value: " <<
  std::cout << "Q7: - Setting kdv1 value" << std::endl;
  kdv1.setKey(666);
  std::cout << "Q7: Is kdv1 value unset (void ?) ? " << std::boolalpha << kdv1.is_void << ". value: " <<

  return 0;
}
```

Running output

```
cd "/home/1laurenj/Ensta/Cours ensta IN204/Session 1/anal_2020/"
g++ -g3 -W -Wall -Wextra main.cpp
./a.out
```

```
## == Testing the 2021 exam correction ==
##
## Q2: kv1 default values ==> {key:0, value: ""}
## Q2: kv1 new value ==> {key:0, value: "This is SPARTAAAA"}
##
## Q3: Are kv1 and kv2 equal ? false
## Q3: Are kv1 and kv2 different ? true
## Q3: kv1 new values ==> {key:1, value: "This is SPARTAAAA"}
## Q3: kv2 new values ==> {key:2, value: "No, this is Patrick !"}
## Q3: kv1 == kv2: false
## Q3: kv1 != kv2: true
## Q3: kv1 >  kv2: false
## Q3: kv1 <  kv2: true
## Q3: kv1 >= kv2: false
## Q3: kv1 <= kv2: true
##
## Q4: kv_templated new values ==> {key:0, value: "1337"}
##
## Q5: [1/4] key: mot, value: 10.000000
## Q5: [2/4] key: le, value: 100.000000
## Q5: [3/4] key: la, value: 80.000000
## Q5: [4/4] key: du, value: 40.000000
##
## Q5: Sorting out the vector by key with the operator<() and std::sort()
## Q5: [1/4] key: mot, value: 10.000000 (sorted by value)
## Q5: [2/4] key: du, value: 40.000000 (sorted by value)
## Q5: [3/4] key: la, value: 80.000000 (sorted by value)
## Q5: [4/4] key: le, value: 100.000000 (sorted by value)
## Q5: Using operator<<() this time: {key:mot, value: 10}
## Q5: Using operator<<() this time: du => 40
## Q5: Using operator<<() this time: {key:la, value: 80}
## Q5: Using operator<<() this time: {key:le, value: 100}
##
## Q7: Is kdv1 value unset (void ?) ? true. value: 0
## Q7: - Setting kdv1 value
## Q7: Is kdv1 value unset (void ?) ? false. value: 666
```