Part I: Flervalgsoppgaver (60 poeng) - LF

Oppgave 1 - intro (5 poeng). Uttrekk, 1 oppgave til hver

1. (5 points) Hvilke (en eller flere) av følgende utsagn er korrekt?
   - √ **C++ støtter objektorientert programmering**
   - ◯ `Templates` kan ikke brukes når man lager generelle klasser
   - ◯ Hvis man trenger kode som kjøres raskt er C++ et dårlig verktøy
   - √ **En typeløs peker (void peker) er det nærmeste C++ kommer en ren maskinadresse**
2. (5 points) Hvilke (en eller flere) av følgende utsagn er korrekt?
   - ◯ C++ støtter ikke objektorientert programmering
   - √ `Templates` **kan brukes når man lager generelle klasser**
   - ◯ Hvis man trenger kode som kjøres raskt er C++ et dårlig verktøy
   - √ **En typeløs peker (void peker) er det nærmeste C++ kommer en ren maskinadresse**
3. (5 points) Hvilke (en eller flere) av følgende utsagn er korrekt?
   - ◯ C++ støtter ikke prosedyreorientert programmering
   - √ `Templates` **kan brukes når man lager generelle klasser**
   - √ **Hvis man trenger kode som kjøres raskt er C++ et godt verktøy**
   - ◯ En typeløs peker (void peker) i C++ er det samme som en `unique_ptr`.
4. (5 points) Hvilke (en eller flere) av følgende utsagn er korrekt?
   - √ **C++ støtter prosedyreorientert programmering**
   - √ `Templates` **kan brukes når man lager generelle klasser**
   - ◯ Hvis man trenger kode som kjøres raskt er C++ et dårlig verktøy
   - ◯ En typeløs peker (void peker) i C++ er det samme som en `unique_ptr`.

Oppgave 2 - Klasser (5 poeng), uttrekk m/1 oppgave til hver

5. (5 points) Hvilke (en eller flere) av følgende utsagn er korrekt?
   - √ **En klasse kan ha flere konstruktører**
   - ◯ Medlemmer i en klasse er default `public`
   - √ `auto`**-typen er spesielt nyttig i generisk kode**
   - √ **En klasse kan ikke ha flere destruktører**
6. (5 points) Hvilke (en eller flere) av følgende utsagn er korrekt?
   - ◯ En klasse kan ikke ha flere konstruktører
   - √ **Medlemmer i en klasse er default** `private`
   - √ `auto`**-typen er spesielt nyttig i generisk kode**
   - √ **En klasse kan ikke ha flere destruktører**
7. (5 points) Hvilke (en eller flere) av følgende utsagn er korrekt?
   - √ **En klasse kan ha flere konstruktører**

- √ **Medlemmer i en klasse er default** `private`
- √ `auto`**-typen er spesielt nyttig i generisk kode**
- ◯ En klasse kan ha flere destruktører

8. (5 points) Hvilke (en eller flere) av følgende utsagn er korrekt?
   - √ **En klasse kan ha flere konstruktører**
   - √ **Medlemmer i en klasse er default** `private`
   - ◯ `auto`-typen er ikke nyttig i generisk kode
   - √ **En klasse kan ikke ha flere destruktører**

Oppgave 3 - This-peker (5 poeng), uttrekk m/1 oppgave til hver

9. (5 points) Hvilke (en eller flere) av følgende utsagn er korrekt?
   - ◯ This-pekeren må slettes i destruktøren
   - ◯ En this-peker kan ikke legges til i en vektor
   - √ **This-pekeren til et objekt er lik pekeren til objektet**
   - √ **Det er ikke mulig å endre på this-pekeren i en medlemsfunksjon**
10. (5 points) Hvilke (en eller flere) av følgende utsagn er korrekt?
    - ◯ This-pekeren må slettes i destruktøren
    - √ **En this-peker kan legges til i en vektor**
    - √ **This-pekeren til et objekt er lik pekeren til objektet**
    - ◯ Det er mulig å endre på this-pekeren i en medlemsfunksjon
11. (5 points) Hvilke (en eller flere) av følgende utsagn er korrekt?
    - √ **Det er ikke nødvendig å slette this-pekeren i destruktøren**
    - ◯ En this-peker kan ikke legges til i en vektor
    - ◯ This-pekeren til et objekt er ikke lik pekeren til objektet
    - √ **Det er ikke mulig å endre på this-pekeren i en medlemsfunksjon**
12. (5 points) Hvilke (en eller flere) av følgende utsagn er korrekt?
    - √ **Det er ikke nødvendig å slette this-pekeren i destruktøren**
    - √ **En this-peker kan legges til i en vektor**
    - ◯ This-pekeren til et objekt er ikke lik pekeren til objektet
    - ◯ Det er mulig å endre på this-pekeren i en medlemsfunksjon

Oppgave 4 og 5 - Uttrekk, alle får 2 oppgaver: finn feil i koden (10 poeng totalt)

13. (5 points) Kodesnutten under handler om terninger (dice), men den inneholder en eller flere feil som gjør at koden ikke vil kjøre/ikke vil kjøre som ønsket dersom `diceValues()`-funksjonen kalles fra f.eks. main.cpp-filen.

```cpp
//The code in the file DiceValues.h
#pragma once
void diceValues(int numberOfDice);
```

```cpp
//The code in the file DiceValues.cpp
#include "DiceValues.h"
#include "std_lib_facilities.h"

vector<int> diceValues(int numberOfDice) {
    vector<int> diceValues;
    constexpr int maxDiceValue = 6;

    for(int i = 0; i < numberOfDice; ++i) {
        int currValue = rand() % maxDiceValue;
        diceValues[i] = currValue;
    }
    return diceValues;
}
```

På hvilke (velg en eller flere) kodelinjer oppstår feilen(e)? Bak hvert kodelinjetall-alternativ står det en parentes som indikerer om linjen er i headerfilen (.h) eller i .cpp-filen (.cpp).

- ◯ 2 (.h)
- √ **3 (.h)**
- ◯ 5 (.cpp)
- ◯ 6 (.cpp)
- ◯ 7 (.cpp)
- ◯ 9 (.cpp)
- √ **10 (.cpp)**
- √ **11 (.cpp)**
- ◯ 13 (.cpp)

14. (5 points)

(a) (2 points) Hvilke (en eller flere) kategori av feil har man gjort om en funksjon som skal generere en tilfeldig bokstav gir en alfakrøll?

- ◯ Type-feil
- ◯ Syntaksfeil
- √ **Logisk feil**
- ◯ Kjøretidsfeil

(b) (3 points) Kodesnutten under inneholder en eller flere feil.

```cpp
#include <iostream>
using namespace std;

int main() {
    int a = 7;
    cout << a / 0;
    return 0;
}
```

Hvor i koden oppstår feilen(e)? Velg en eller flere kodelinjer:

◯ 1
◯ 2
◯ 4
◯ 5
√ **6**
◯ 7

15. (5 points) Kodesnutten under inneholder en eller flere feil.

```
#include <iostream>
using namespace std;

int main() {
        cout << isOdd(2);
        return 0;
}

bool isOdd(int num) {
  if (num % 2)
  {
    return false;
  }
  return true;
}
```

Hvilke (en eller flere) av alternativene under fungerer som ønsket?

√
```
#include <iostream>
using namespace std;

bool isOdd(int num);

int main() {
        cout << isOdd(2);
        return 0;
}

bool isOdd(int num) {
  if (num % 2)
  {
    return true;
  }
  return false;
}
```

◯
```
#include <iostream>
using namespace std;

int main() {
        cout << isOdd(2);
        return 0;
}

bool isOdd(int num) {
  if (num % 2)
  {
    return true;
  }
  return false;
}
```

√
```
#include <iostream>
using namespace std;

bool isOdd(int num) {
  if (num % 2 == 0)
  {
    return false;
  }
  return true;
}

int main() {
        cout << isOdd(2);
        return 0;
}
```

◯
```
#include <iostream>
using namespace std;

bool isOdd(int num) {
  if (num % 2)
  {
    return false;
  }
  return true;
}

int main() {
        cout << isOdd(2);
        return 0;
}
```

16. (5 points) Kodesnutten under inneholder en eller flere feil.

```
enum class Color{Red = 0, Blue = 1, Yellow = 2};

bool isYellow(Color 2) {
    return c == 2;
}
```

Hvilke (en eller flere) av alternativene under fungerer som ønsket?

◯
```
enum class Color{Red = 0, Blue = 1, Yellow = 2};

bool isYellow(Color c) {
    return c == 2;
}
```

◯
```
enum class Color{Red = 0, Blue = 1, Yellow = 2};

bool isYellow(Color 2) {
    return c == Color::Yellow;
}
```

√
```
enum class Color{Red = 0, Blue = 1, Yellow = 2};

bool isYellow(Color c) {
    return c == Color::Yellow;
}
```

◯
```
enum class Color{Red = 0, Blue = 1, Yellow = 2};

bool isYellow(Color c) {
    return c == Yellow;
}
```

Oppgave 6 - Templates (10 poeng)

Nedtrekksmeny med ulike alternativ.

17. (10 points) Se på koden under.

```
#include "std_lib_facilities.h"

class MyStack {
public:
    vector<int> vec;

    void push(int number);

    void pop();
};

inline void MyStack::push(int number) {
    vec.push_back(number);
}

inline void MyStack::pop() {
    vec.pop_back();
}
```

Vi ønsker å gjøre stakken om fra noe som bare håndterer heltall til noe som håndterer ulike typer ved å bruke `templates`. Hvilke (en eller flere) av linjene i koden må vi endre i klassen for å gjøre dette? `inline`-nøkkelordet er brukt for å ha alt i headerfilen.

Solution:

```cpp
#include "std_lib_facilities.h"
template<typename T>
class MyStack {
public:
    vector<T> vec;

    void push(T number);  //evt. med const er ok

    void pop();
};

template<typename T>
void MyStack<T>::push(T number) {   //evt. med const hvis const i 7, er ok
    vec.push_back(number);
}
template<typename T>
void MyStack<T>::pop() {
    vec.pop_back();
}
```

Oppgave 7 - Arv (10 poeng) - uttrekk, 1 oppgave til hver

18. (10 points) Hvilke (en eller flere) av de følge påstandene er korrekte?

```
#include "std_lib_facilities.h"

class Cat {
protected:
    string name;
    int age;
public:
    Cat(string name, int age):
      name{name}, age{age} {}

    virtual void meow() {cout << "Meow\n";}
    Cat& operator=(const Cat&) = delete;

    ~Cat() {}
};


class NorwegianCat : public Cat {
public:
    NorwegianCat(string name, int age = 12) :
      Cat{name, age} {}
    void meow() override {cout << "Meow (in  Norwegian)\n";}
};
```

- √ **NorwegianCat har tilgang til medlemsvariablene i Cat-klassen.**
- √ **Funksjonen `meow()` i NorwegianCat er virtual.**
- ◯ Cat er en abstrakt klasse
- √ **Funksjonen `meow()` i Cat er virtual.**

19. (10 points) Hvilke (en eller flere) av de følgende linjene i klassen kan du endre på for å gjøre klassen `Rock` abstrakt samtidig som begge klassene `Taconite` og `Granite` forblir konkrete?

```
1  class Rock {
2  public:
3      double weight;
4      Rock(const Rock&) = delete;
5      Rock& operator = (const Rock&) = delete;
6
7      virtual bool isEatable() {return false;}
8      int getWeight() {return weight;}
9      Rock(double weight) : weight{weight} {}
10 };
11
12 class Taconite : public Rock {
13 public:
14     Taconite(double weigth) : Rock{weigth} {}
15     bool isEatable() override {return true;}
16 };
17
18 class Granite : public Rock {
19 public:
20     Granite(double weigth) : Rock{weigth} {}
21     bool isEatable() override {return false;}
22 };
```

- ◯ 2
- ◯ 4
- √ **7**
- ◯ 8
- ◯ 12
- ◯ 14
- ◯ 15
- ◯ 18
- ◯ 20
- ◯ 21

20. (10 points) Hvilke (en eller flere) av de følge påstandene er korrekte?

```
class Person {
private:
    int age;
    string name;
public:
    Person(string name, int age) : age{age}, name{name} {}
    virtual void working() = 0;
};

class Worker : public Person{
private:
    string workPlace;
public:
    Worker(string name, int age, string workPlace) :
          Person{name, age}, workPlace{workPlace} {}
    void working() override {cout << "C++ developer at " + workPlace;}
};
```

- √ `Person` **er en abstrakt klasse.**
- ◯ `Worker` er en abstrakt klasse.
- √ **Funksjonen** `working()` **i** `Person`**-klassen er virtuell.**
- √ **Funksjonen** `working()` **i** `Worker`**-klassen er virtuell.**

Oppgave 8 - Dynamisk minne del 1 (5 poeng) - uttrekk (1 oppgave til hver)

21. (5 points) Hvilke (en eller flere) av alternativene er nødvendige når det brukes dynamisk minne i en klasse?

- √ **Destruktør**
- √ **Konstruktør**
- ◯ Grunn kopiering
- √ **Operatoroverlasting av** `operator=`

22. (5 points) Hvilke (en eller flere) av alternativene er nødvendige når det brukes dynamisk minne i en klasse?

- √ **Destruktør**
- √ **Konstruktør**
- √ **Dyp kopiering**
- ◯ Operatoroverlasting av `operator<<`

23. (5 points) Hvilke (en eller flere) av alternativene er nødvendige når det brukes dynamisk minne i en klasse?

- √ **Destruktør**
- ◯ Delegerende konstruktør
- √ **Dyp kopiering**
- √ **Operatoroverlasting av** `operator=`

Oppgave 9 - Dynamisk minne del 2 (5 poeng) (uttrekk, 1 oppgave til hver)

24. (5 points) Gitt klassedeklarasjonen for `Hometown` under.

```
class Hometown{
    string name;
public:
    Hometown(string name) : name{name} {}
};
```

I hvilke (en eller flere) av klassedeklarasjonene under kan det være lurt å implementere kopikonstruktøren selv?

◯
```
class Person {
        string name;
        Hometown home;
    public:
        Person(string name, string city) : name{name}, home{city} {}
        Person& operator=(Person&) = delete;
};
```

√
```
class Person {
        string name;
        Hometown * home = nullptr;
    public:
        Person(string name, string city) : name{name} {home = new Hometown(city);}
};
```

◯
```
class Person {
        string name;
        Hometown home;
    public:
        Person(string name, string city) : name{name}, home{city} {}
        Person() = delete;
};
```

√
```
class Person {
    string name;
    Hometown * home = nullptr;
  public:
    Person(const Person & Me) = default;
    Person(string name, string city) : name{name} {home = new Hometown(city);}
  };
```

25. (5 points) Se på klassedeklarasjonene.

```
struct Leash {
    int length;
};

class Dog{
private:
    string name;
    Leash* dogLeash = nullptr;
public:
    Dog(string dogName, int leashLength) {
        name = dogName;
        dogLeash = new Leash{leashLength};
    }
};
```

Hvilke (en eller flere) av de alternative tilordningsoperatorene nedenfor gjør en riktig dyp kopi? Du kan anta at både destruktøren og kopikonstruktøren til klassen Dog er implementert korrekt.

◯
```
Dog& Dog::operator=(Dog& rhs) {
    std::swap(name, rhs.name);
    std::swap(dogLeash, rhs.dogLeash);
    return *this;
  }
```

√
```
Dog& Dog::operator=(Dog rhs) {
    std::swap(name, rhs.name);
    std::swap(dogLeash, rhs.dogLeash);
    return *this;
  }
```

◯
```
Dog& Dog::operator=(Dog& rhs) {
    name = rhs.name;
    dogLeash = rhs.dogLeash;
    return *this;
  }
```

◯
```
Dog& Dog::operator=(Dog rhs) {
    name = rhs.name;
    dogLeash = rhs.dogLeash;
    return *this;
  }
```

Oppgave 10 - Pekere og iteratorer (5 poeng) - uttrekk, 1 oppgave til hver

26. (5 points) Gitt følgende kodesnutt:

```
#include "std_lib_facilities.h"

int main(){
    vector<int> myVec{2, 5, 42, 8};
    auto iter = myVec.begin();
    vector<int>* ptr = &myVec;
    return 0;
}
```

Hvilke (en eller flere) av disse påstandene stemmer?

- √ `(*ptr).at(2)` **og** `ptr->at(2)` **betyr det samme.**
- ◯ `iter` og `myVec.rend()` peker på samme element.
- √ `iter[3]` **og** `*(myVec.end()-1)` **betyr det samme.**
- ◯ `ptr` og `iter` peker på det samme.

27. (5 points) Gitt følgende kodesnutt:

```
#include "std_lib_facilities.h"

int main(){
    vector<int> myVec{2, 5, 42, 8};
    auto iter = myVec.begin();
    vector<int>* ptr = &myVec;
    return 0;
}
```

Hvilke (en eller flere) av disse påstandene stemmer?

- √ `(*ptr).at(2)` **og** `ptr->at(2)` **betyr det samme.**
- √ `iter` **og** `myVec.rend()` **peker på forskjellige element.**
- ◯ `iter[3]` og `*(myVec.end()-1)` betyr *ikke* det samme.
- ◯ `ptr` og `iter` peker på det samme.

28. (5 points) Gitt følgende kodesnutt:

```
#include "std_lib_facilities.h"

int main(){
    vector<int> myVec{2, 5, 42, 8};
    auto iter = myVec.begin();
    vector<int>* ptr = &myVec;
    return 0;
}
```

Hvilke (en eller flere) av disse påstandene stemmer?

- ◯ `(*ptr).at(2)` og `ptr->at(2)` betyr *ikke* det samme.
- √ `iter` **og** `myVec.rend()` **peker på forskjellige element.**
- ◯ `iter[3]` og `*(myVec.end()-1)` betyr *ikke* det samme.
- √ `ptr` **og** `iter` **peker *ikke* på det samme.**

29. (5 points) Gitt følgende kodesnutt:

```cpp
#include "std_lib_facilities.h"

int main(){
    vector<int> myVec{2, 5, 42, 8};
    auto iter = myVec.begin();
    vector<int>* ptr = &myVec;
    return 0;
}
```

Hvilke (en eller flere) av disse påstandene stemmer?

- ◯ `(*ptr).at(2)` og `ptr->at(2)` betyr *ikke* det samme.
- √ `iter` **og** `myVec.rend()` **peker på forskjellige element.**
- √ `iter[3]` **og** `*(myVec.end()-1)` **betyr det samme.**
- ◯ `ptr` og `iter` peker på det samme.