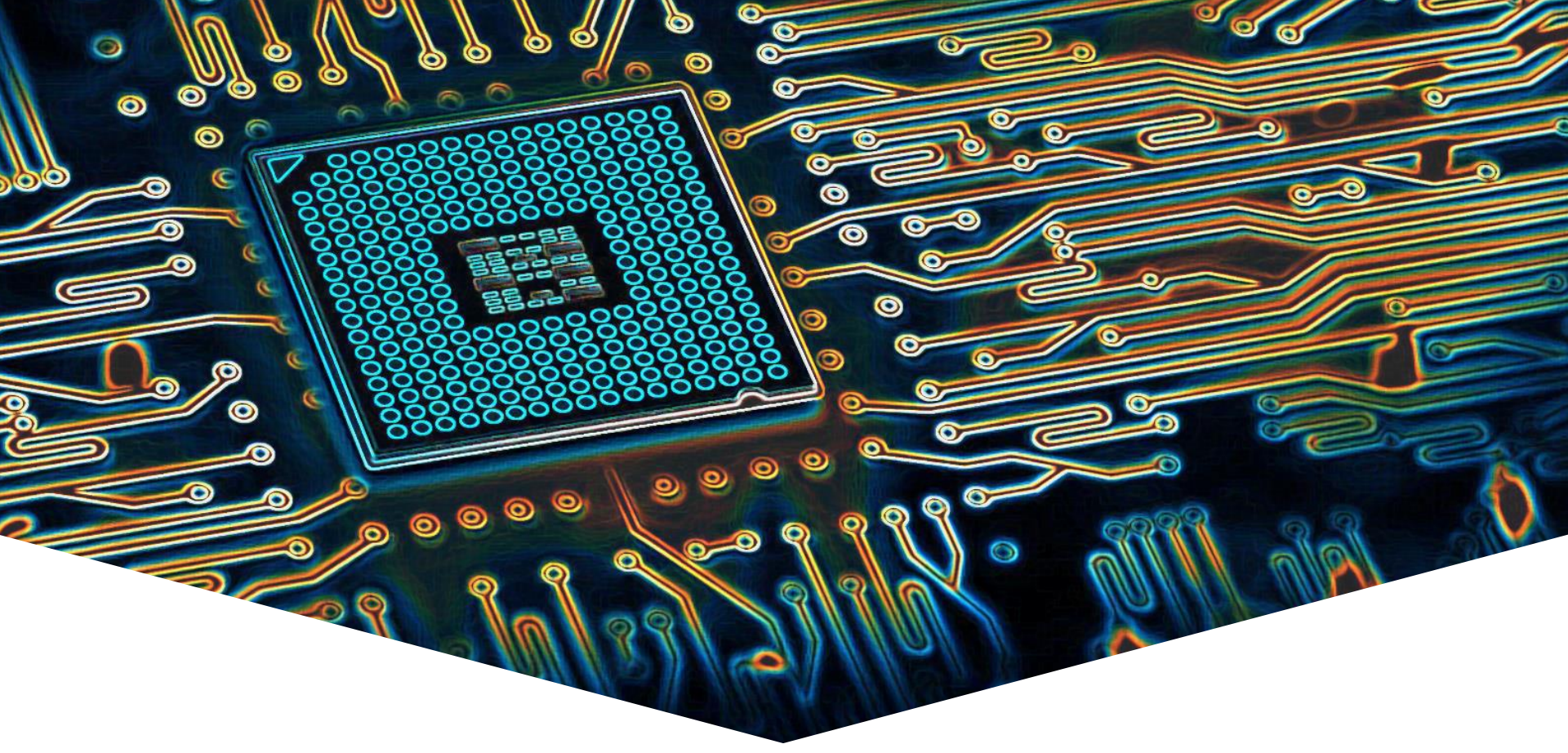
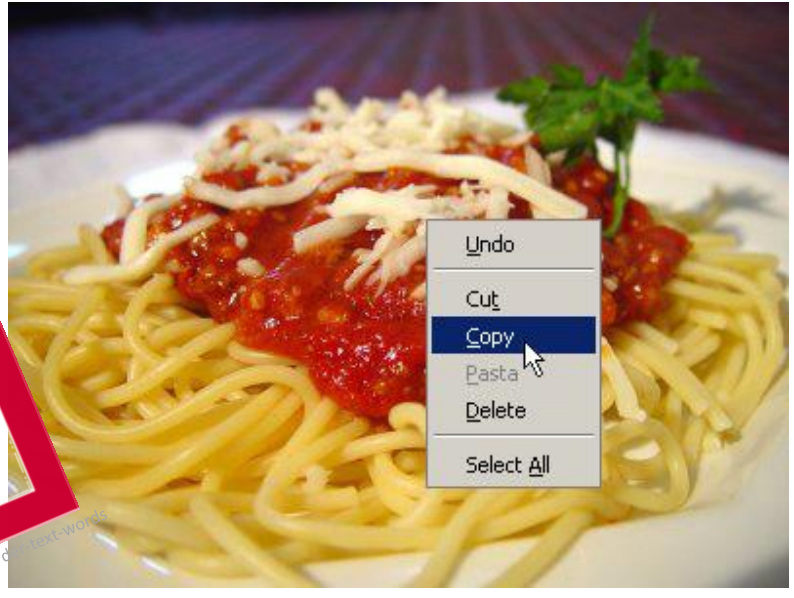


EMS30 Week 6 Les 1: Templates

Herbruikbaarheid?


```
void swap(int& p, int& q) {
    int t {p};
    p = q;
    q = t;
}

// ...
    int i {3};
    int j {4};
    swap(i, j);
// ...
```



Bron: https://me.me/i/undo-cut-copy-pasta-delete-select-all-it-is-delicious-22661521

- Wat te doen als we twee variabelen van het type `double` willen verwisselen?
- Wat te doen als we twee variabelen van het type `Breuk` willen verwisselen?

DRY: Don't Repeat Yourself

Bron: https://www.magicalquote.com/seriesquotes/listen-very-carefully-i-shall-say-this-only-once/

Generieke functie

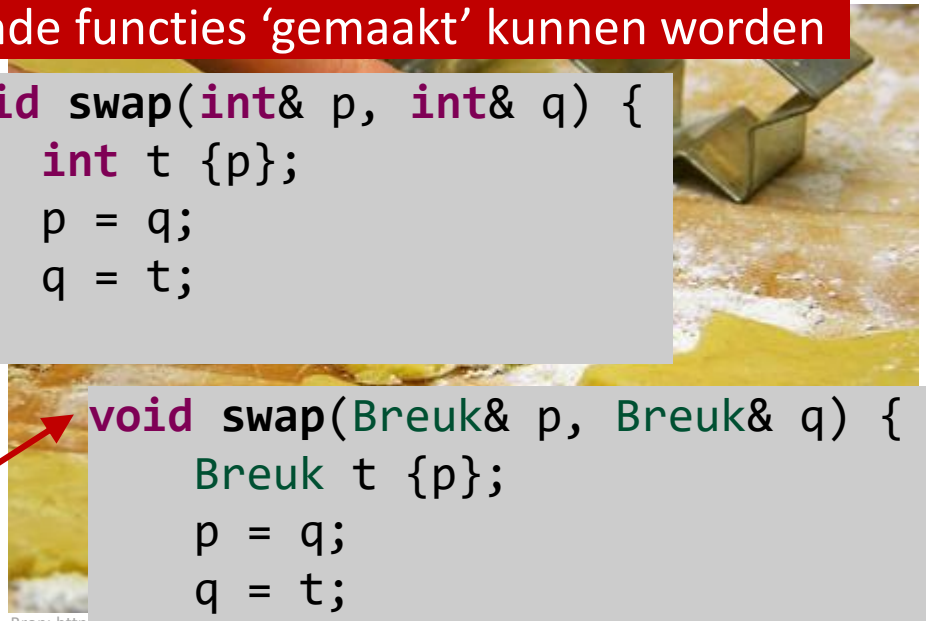
Gebruik een **functietemplate**:

```
template <typename T> void swap(T& p, T& q) {
    T t {p};
    p = q;
    q = t;
}

//  ...
    int i {3};
    int j {4};
    swap(i, j);
//  ...
    Breuk b {1, 2};
    Breuk c {3, 4};
    swap(b, c);
```

Een functietemplate is een '**mal**' waarmee verschillende functies 'gemaakt' kunnen worden

```
void swap(int& p, int& q) {
    int t {p};
    p = q;
    q = t;
}
```



```
void swap(Breuk& p, Breuk& q) {
    Breuk t {p};
    p = q;
    q = t;
}
```

Bron: http

Class Dozijn

In de **class Dozijn** kun je 12 integers opslaan.

```
class Dozijn {
public:
    void zet_in(int index, int waarde);
    int lees_uit(int index) const;
private:
    int data[12];
};

//  ...
    Dozijn d;
    d.zetIn(3, 13);
//  ...
    cout << "De plaats nummer 3 in d bevat de waarde: "
         << d.leesUit(3) << endl;
```

Bron: https://cdn.pixabay.com/photo/2019/09/16/23/28/eggs-4482186_960_720.jpg

Class Dozijn


```
void Dozijn::zet_in(int index, int waarde) {
    if (index >= 0 && index < 12)
        data[index] = waarde;
}

int Dozijn::lees_uit(int index) const {
    if (index >= 0 && index < 12)
        return data[index];
    return 0; /* ik weet niets beters */
}
```

- Wat te doen als we twaalf variabelen van het type `double` willen opslaan?
- Wat te doen als we twaalf variabelen van het type `Breuk` willen opslaan?

Generieke class

Gebruik een **class template**:

```
template<typename T> class Dozijn {
public:
    void zet_in(int index, const T& waarde);
    const T& lees_uit(int index) const;
private:
    T data[12];
};

//  ...
    Dozijn<int> di;
```

Een class template is een '**mal**' waarmee verschillende classes 'gemaakt' kunnen worden

```
class Dozijn {
public:
    void zet_in(int index, const int& waarde);
    const int& lees_uit(int index) const;
private:
    int data[12];
};
```

Generieke class Dozijn


```
template<typename T>
void Dozijn<T>::zet_in(int index, const T& waarde) {
    if (index >= 0 && index < 12)
        data[index] = waarde;
}

template<typename T>
const T& Dozijn<T>::lees_uit(int index) const {
    if (index < 0)
        index = 0;
    if (index > 11)
        index = 11;
    return data[index];
}
```

Generieke class Dozijn


```
Dozijn<string> provincies;
provincies.zet_in(0, "Drenthe");
provincies.zet_in(1, "Flevoland");
// ...
provincies.zet_in(10, "Zeeland");
provincies.zet_in(11, "Zuid-Holland");
cout << "provincies.get_uit(2) = "
     << provincies.get_uit(2) << '\n';
```


- Wat als we doen als we meer/minder dan twaalf elementen willen opslaan?

Bron: https://nl.wikipedia.org/wiki/Bestand:NederlandseProvinciesLarge.png

Generieke class

Gebruik een tweede **template** parameter:

```
template<typename T, size_t N> class Rij {
public:
    void zet_in(size_t index, const T& waarde);
    const T& lees_uit(size_t index) const;
    constexpr size_t aantal_plaatsen();
private:
    T data[N];
};

//  ...
    Rij<char, 26> alfabet;
```

Een class template is een '**mal**' waarmee verschillende classes 'gemaakt' kunnen worden

```
class Rij {
public:
//  ...
private:
    char data[26];
};
```

Template class `std::array`

- `std::array<T, N>` in C++ vervangt de C array.
- Het aantal elementen `N` ligt vast na het compileren.
- Elementen kunnen worden opgevraagd met `operator[]`.
- Je kunt een `std::array` 'gewoon' vergelijken, toekennen en kopiëren.
- Je kunt een `std::array` element voor element doorlopen met een range-based for.
- `std::array` heeft memberfuncties:
  - `size()` geeft het aantal elementen
    (type: `std::array<T, N>::size_type`).
  - `at(`*n*`)` geeft reference naar element *n*.
    Geeft een fout (exception) als element *n* niet bestaat.

Template class std::array


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

int main() {
    // definieer array van 15 integers
    array<int, 15> a;
    // vul met kwadraten
    int i {0};
    for (auto& e: a) {
        e = i * i;
        ++i;
    }
    // druk af
    for (auto e: a) {
        cout << e << " ";
    }
```

```
0 1 4 9 16 25 36 49 64 81 100 121 144 169 196
```

Template class `std::vector`

- `std::vector<T>` in C++ is een **dynamische** array.
- De `std::vector` kan groeien en krimpen.
- Elementen kunnen worden opgevraagd met `operator[]`.
- Je kunt een `std::vector` "gewoon" vergelijken, toekennen en kopiëren.
- Je kunt een `std::vector` element voor element doorlopen met een range-based for.
- `std::vector` heeft memberfuncties:
  - `size()` geeft het aantal elementen
    (type: `std::vector<T>::size_type`).
  - `at(`*n*`)` geeft reference naar element *n*.
    Geeft een fout (exception) als element *n* niet bestaat.
  - `push_back(`*e*`)` voeg element *e* aan de vector toe.

Template class std::vector

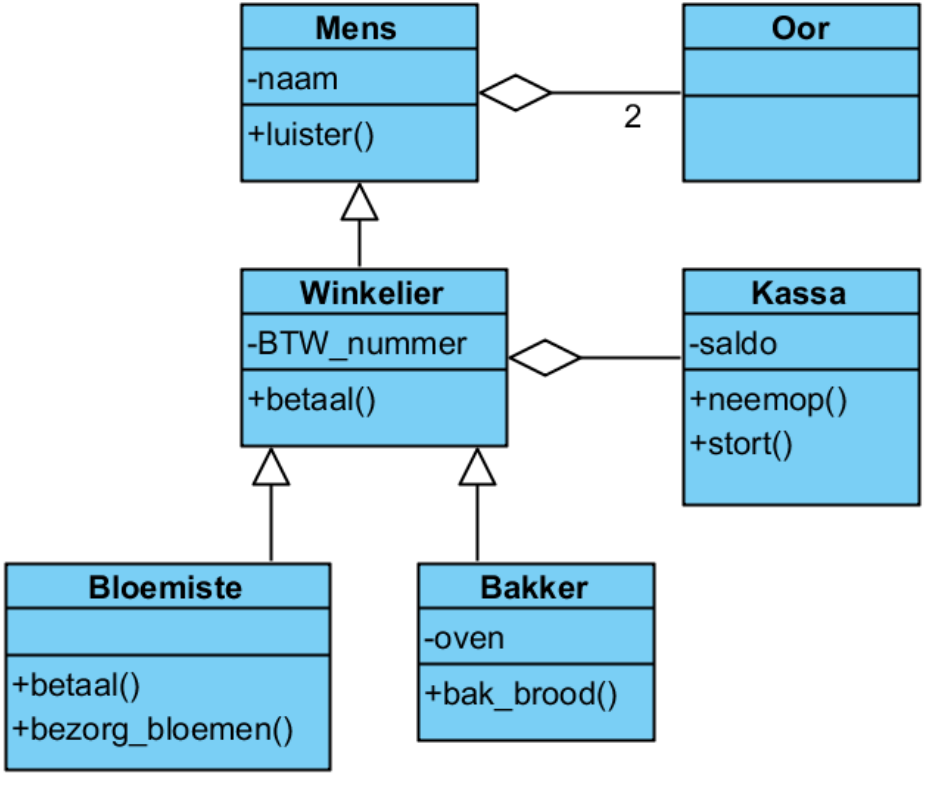

```
#include <iostream>
#include <vector>
using namespace std;

int main() {
    // definieer vector van integers
    vector<int> v;
    int aantal;
    cin >> aantal;
    // vul met kwadraten
    for (int i {0}; i < aantal; ++i) {
        v.push_back(i * i);
    }
    // druk af
    for (auto e: v) {
        cout << e << " ";
    }
```

```
11
0 1 4 9 16 25 36 49 64 81 100
```

Volgende les…

Overerving



Aan de slag!

Aan de slag met [Opdrachten_Week_6_Les_1.pdf]