Concepten C++

Basisconcepten C++

Conversies

```
double d; int i;
i = (int)(d*2+3);
i = int(d*2+3);
```

Standaardstrings

```
string s1; s1 = "Hij zei: \"Hallo\"\n";
// \" zorgt ervoor dat we " kunnen printen
string s2("test"); string s3 = s2;
string s4(s3); string s5(10,'c');

//concatenatie
string s = "dag"; s = s + " Jan"; s = s + '!';
//vergelijken van strings
if (s < "woord") // s komt alfabetisch voor "woord"
if (s >= t) // is s alfabetisch groter dan/gelijk aan t
if (s == "stop") // bevat s het woord "stop"

// lengte bepalen
string s = "dag"; int l = s.size(); l = s.length() // 3
// string ontleden
for (char c: s) cout << c;
```

Default parameters

```
void schrijf_lijn(char c = '-', int i = 80);
```

References

```
void wissel(int &, int &);
int main() {
      int a = 5, b = 6; wissel(a,b);
}
void wissel(int &x, int &y) {
      int h = x; x = y; y = h;
}
```

Functie-templates

```
template <typename T> //template prefix
void wissel(T &a, T &b) {
      T hulp = a; a = b; b = hulp;
} // enkel types die alle operatoren ondersteunen
// kunnen opgeroepen worden als T, dus geen structs
```

Uitvoer-invoer

```
//uitvoer
cout << ... << endl;
//invoer
cin >> var;
//string
getline(cin, var);
// char
var = cin.get(); var = getchar();
```

Werken met bestanden

```
ifstream inv; //invoer
ofstream uitv; //uitvoer
inv.open(bestandsnaam); uitv.open(bestandsnaam, ios::...);
//controle op openen
if (inv.is_open()) - if (!inv.fail()) - if (inv)
//lezen/schrijven
inv >> var; var = inv.get(); getline(inv, var);
uitv << var << " " << var << " " << endl;
//sluiten
inv.close(); uitv.close();
```

Dynamisch geheugenbeheer

```
//array
int *r = new int[grootte]; delete[] r;
//variabele
int *a = new int; delete a;
```

Nieuw sinds C++11 (1)

Functies als parameters

```
#include <functional>
bool zoek(const string *t, int n, function<bool (const string &)> func)
{...});
function<returntype (prototypelijst)> naam {body}
```

Lambda functies

```
[captures] (parameterlijst) -> returntype {statements}
//-> returntype moet niet vermeld worden
```

Nullptr

```
// vervangt NULL en 0
nullptr = int *;
```

Smart pointers

```
unique_ptr<type> var; // nullptr
var = make_unique<type>(inhoud);
//kopiëren gaat niet, enkel .move() en .swap()
unique_ptr<type> var2 = make_unique<type>(inhoud);
var2.swap(var); // wisselen ze
var2 = move(var) // transfereert
var2.reset() // geheugen vrijgeven

shared_ptr<type> var, var2;
var = make_shared<type>(inhoud);
var2 = var; // beide zijn eigenaar
```

Collections

Iteratoren

```
[typename] collectiontype::[const_]iterator naam_var;
naam_var = var.begin(); naam_var = var.end();
//vooruit en achteruit
++naam_var; naam_var++; --naam_var; naam_var++;
//verplaatsen
advance(naam_var, aantal);
```

Vector

```
vector<type> var(grootte); vector<type> var(grootte, inhoud);
var.size() --> grootte vector
var.capacity() --> capaciteit vector
```

[unordered_]set

Duplicaten worden hier vermeden, elk element kan dus maar één keer voorkomen. Afhankelijk van unordered_ of niet zal alles gesorteerd worden. -> moeten wel types zijn die vergeleken kunnen worden.

[unordered_]multiset

Hetzelfde als de set, maar hier kunnen duplicaten wel voorkomen.

OGP in C++

Klassen

```
class voorbeeld {
	public:
		void set_a(int = waarde); // setter
		int get_a() const; // getter, const is belangrijk!!
		voorbeeld(int i, int j) : a(i), b(j) {} // constructor
		~voorbeeld() {} // destructor
	private:
		int a,b; // attributen
};
```

Friend

```
class Y { ... };
class Z {
	friend class Y; // Y kan nu aan alle data van Z, niet omgekeerd
	...
};
```

Klasse-templates

```
template <typename T>
class voorbeeld {
	public:
		voorbeeld();
		voorbeeld(const T&, const T&);
		T get_iets() const;
		void set_iets(const T&);
		voorbeeld<T> operator+(const voorbeeld<T> &) const;
	private:
		T var, var2;
};
```

Nieuw sinds C++11 (2)

Auto

```
auto a = 0; // int
auto b = 'a'; // char
// geen goed idee om auto voor 'eenvoudige' types te gebruiken
auto it = verzameling.begin();
// beter voor complexe structuren zoals iteratoren, etc.
```

Nieuwe initialisatiesyntax

```
class A {
	int x = 7;
	int y{9};
	public:
		A() {}
};
```

Move constructor en operator

```
A(const A &); // copy
A(A &&); // move
A& operator=(const A&); // toekenningsop.
A& operator=(A &&); // move operator
```

Exception handling

```
// exceptie opwerpen
throw "exceptie: hier staat iets!";
// exceptie opvangen
catch (const char *s) {cout << s << endl;} // C-string
catch (...) {cout << "ai" << endl;} // alle excepties, altijd laatst!
// belangrijkste klassen
logic_error
runtime_error
// opvangen
catch (const runtime_error &e) {cout << e.what() << endl;}
// zelfgemaakte klasse
class mijn_error : public runtime_error {
	public:
		mijn_error(): runtime_error("mijn melding") {}
		mijn error(const string &what) : runtime_error(what) {}
};
```