-- C++ CHEATSHEET --

Headers

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

Compilation/Execution

```
$ g++ test.cc -o test -Wall -std=c++11
$ ./test
```

Commentaires

```
// ...          /* ... */
```

tout ce qui suit // ou est entre /* et */ est un commentaire

Fonctions

```
typeout  funcName(typein_1  attr1, typein_2  attr2) {
        // ...
        return  value;
}
```

Types

char	un seul caractère (1 byte)
bool	valeur booléenne : *true* ou *false* (1 byte)
int	nombre entier positif ou négatif (4 bytes)
float	nombre réel (4 bytes)
double	nombre réel (8 bytes)
intX_t	nombre entier positif ou négatif (X bits, X={8,16,32,64})
uintX_t	nombre entier sans signe (X bits, X={8,16,32,64})

Variables

int i(42); ou int i = 42;	déclaration
i = 5;	assignation
i	valeur
&i	adresse mémoire

Pointeurs

int *p(0);	déclaration
p = &i;	assignation
int *p = new int(42);	déclaration + assignation
delete p;	destruction
p	valeur (adresse de la variable pointée)
&p	adresse mémoire
*p	valeur de la variable pointée

Conditions

a==b	vrai si a égal b
a !=b	vrai si a différent de b
a>b	vrai si a strictement supérieur à b
a>=b	vrai si a supérieur ou égal à b
a %b	vrai si a proportionnel à b

Opérateurs logiques

A || B	vrai si A ou B
A && B	vrai si A et B
A ^ B	vrai si A ou B mais pas les deux (xor)

Entrée/Sortie

std::cout<< "a" ;	affiche a sur la sortie standard
std::cout<< a ;	affiche la valeur de a sur la sortie standard
<<std::endl ;	affiche un retour à la ligne sur la sortie standard
std::cin>>a;	enregistre l'entrée standard dans la variable a

Structures de contrôle

```
if (cond1) {
      // ...
}
else if(cond2) {
      // ...
}
else {
      // ...
}
```

```
for (i=0; i<size; i++)
{

      // ...
}
```

```
while (condition) {
      // ...
}
```

Opérateurs de boucle

`break;`	arrête la boucle courante (`for` ou `while`)
`continue;`	saute à l'itération de boucle suivante (`for` ou `while`)

Smart pointers

`#include <memory>`	include
`unique_ptr<int>ptr= make_unique<int>(5);`	déclaration (unique)
`shared_ptr<int>ptr1= make_shared<int>(5);`	déclaration (shared)
`shared_ptr<int>ptr2(ptr1) ;`	déclaration (shared)

Tableaux

`int tab[5];`	tableau statique
`int *tab;`	tableau "dynamique", déclaration
`tab = new int[5];`	tableau "dynamique", allocation
`delete[] tab;`	tableau "dynamique", désallocation

Vecteurs

`#include <vector>`	include
`vector<int> vec(5);`	déclaration
`vec.push_back(val);`	ajoute l'élément `val` au vecteur
`vec.pop_back();`	supprime le dernier élément du vecteur
`vec.size();`	taille du vecteur

Itérateurs

`vector<int>::iterator it;`	déclaration d'un itérateur
`vec.begin();`	itérateur sur le premier élément de `vec`
`vec.end();`	itérateur sur le dernier élément de `vec`

Itérations

`for(int *it=tab; it!=tab+5; it++)`	tableaux
`for(it=vec.begin(); it!=vec.end(); ++it)`	vecteurs (`it` prédéclaré)
`for(auto it=vec.begin(); it!=vec.end(); ++it)`	vecteurs (déclaration à la volée)

Classes

`class ClassName`	définition
`{`	
`  private:`	
`    type1 attrib;`	attribut
`  public:`	
`    classname(...);`	constructeur
`    ~classname(...);`	destructeur
`    type2 method(...);`	méthode
`} ;`	
`ClassName::ClassName(...) {}`	implémentation constructeur
`type2 ClassName::method(...) {}`	implémentation méthode

Surcharge d'opérateur

`<typeout> operatorX(<type> const&, <type> const&);` surcharge de `X` pour les objets `<type>`