Hello World

```
#include <iostream>

using namespace std;

int main() {
   cout << "Hello␣World" << endl;
   return 0;
}
```

Types fondamentaux

`bool`	8 bits
`char` `unsigned char`	8 bits
`short`	16 bits
`int` `unsigned int`	$\geq$16 bits
`long`	$\geq$32 bits
`long long`	64 bits
`float`	32 bits
`double`	64 bits

Les chaînes de caractères :

```
#include <string>
```

Possibilité de concaténer, de comparer, d'utiliser la méthode `size()` etc.

Déclaration et initialisation

```
int nombre;
int nombre = 0;
int nombre(0);
char lettre = 'a';
string chaine = "Bonjour";
```

Les fonctions

```
type nomFonction(type1 argument1, type2 argument2) {
...
return ... //sauf si type = void
}
```

Les références

```
int nombre = 0;
int& alias = nombre;
```

Passage par référence $\neq$ Passage par valeur

```
void permuter(&x,&y){
    int temp = x;
    x = y;
    y = temp;
}
int main()
{
    int a=1, b=2;
    permuter(a,b);
    // a=2 et b=1
    return 0;
}
```

Les opérateurs

```
+, -, *, /, %, +=, -=, *=, /=, %= //arithmetic
&, |, ^, ~, <<, >> //bitwise
sizeof()

int y, x=3;
//post-incrementation
y = x++;
//pre-incrementation
y = ++x;

//operateur ternaire ?
// condition ? resulat_vrai : resultat_faux;
x = (y>0) ? 1 : -1;
```

Cast

```
float f = 3.2;
int n;
//C-like casting
n = (int) f;
//functional casting
n = int (f);
```

cmath

```
#include <cmath>
pow();
sqrt();
exp();
log(); //ln()
log10();
sin();
cos();
tan();
```

Condition

```
==, !=, >, <, >=, <= //comparison
!, &&, || //logical

bool test = true;
if (test) { // if (test == True) {
...
}
else if (condition) {
...
}
else {
...
}
```

Condition switch

```
switch (variable)
{
    case valeur1:
        ...
        break;

    case valeur2:
        ...
        break;

    default:
        ...
        break;
}
```

Boucles

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

do {
...
} while (condition);

// (initialisation ; condition ; incrementation)
for (int i=0 ; i<=10 ; i++) {
...
}
```

Les fichiers

Déclaration ou **prototype** des fonctions dans un fichier d'en-tête (fichier.h)

```
#ifndef FICHIER_H
#define FICHIER_H
void f1(arg1, arg2=0);
//valeurs par defaut seulement dans la declaration
#endif
```

Définition des fonctions dans un fichier.cpp

Les tableaux statiques

```
int const taille = 10;
//la taille doit etre une constante
int tab[taille];
//Parcours du tableau
for(int i(0); i<taille; ++i)
{
    cout << tab[i] << endl;
}
//Avec une fonction
void fonction(double tableau[], int taille) {
    //impossible de retourner un tableau statique
}
//tableau multidimensionnel
int matrice[taille][taille];
```

Gestion des fichiers

```
#include <fstream>
//Ecriture
ofstream fic("fichier.txt", ios_base::app);
int a = 2;
if (fic.is_open()) {
    fic << a << endl;
    fic.close(); //facultatif
}
//Lecture
ifstream fic("fichier.txt");
if (fic.is_open()) {
    string line;
    while (getline(fic,line)) {
        cout<<line<<endl;
    }
}
```

Les pointeurs

```
// declaration
int* ptr = 0;
//adresse de n
int n = 5;
ptr = &n;
//dereferencement
cout << *ptr << endl;
```

Les classes

Un objet est une instance d'une classe

```
//Forme de Coplien
class T {
    attribut1;
    attribut2;
public :
    T(); //constructeur par defaut
    T (const T&); //constructeur par recopie
    ~T(); //destructeur
    T& operator=(const T&);
    //operateur d'affectation

    T(type1 arg1, type2, arg2);
    void method();
};
```

Liste d'initialisation

- Dans le même ordre que la déclaration des attributs
- Obligatoire pour les attributs constants
- A privilégier si certains attributs sont des objets

```
T::T(int i, double x):attribut1(i),attribut2(x) {}
```

Encapsulation

Pour les membres (attributs et méthodes) d'une classe

- **private** (par défaut) : le nom du membre n'est connu que des fonctions membres et des fonctions amies de la classe
- **protected** : idem que private + le membre est accessible à toutes les méthodes des classes dérivées
- **public** : accessible par tous

Accesseurs et mutateurs

Accesseur

```
int T::getAttribut() const {
    return attribut;
}
```

Mutateur

```
void T::setAttribut(int var) {
    attribut = var;
}
```

Gestion de la mémoire

```
T *ptr1 = new T;
ptr1->method(); //*ptr1.method();
delete prt1;
int* ptr2 = new int[10];
delete[] ptr2;
```

Dans une classe, `this` est un pointeur sur l'objet :

```
void T::method() {
    this->attribut = 3;
}
```

Amitié

- Fonction **non-membre** de la classe
- Accès à tous les membres (public ou private)
- Mot réservé `friend`
- Possibilité de déclarer une classe amie

```
class T {
    friend void fonc_amie();
};
```

Les membres static

- Partagés par tous les objets de la classe
- Un seul exemplaire par classe
- Attribut static ou fonction membre static
- Mot réservé `static`

```
class T {
    static int nbObj;
};
```

Initialisation dans .cpp

```
int T::nbObj = 0;
```

Les membres constants

```
class T {
    int attribut;
    public:
        void fnonconst();
        void fconst() const;
};
void T::fconst() const {
    attribut = 0 //INTERDIT
}
int main() {
    T x;
    const T y;
    x.fnonconst() //OK
    x.fconst() //OK
    y.fnonconst() //INTERDIT
    y.fconst() //OK
```

Passage par référence constante

Evite la copie et empêche la modification

```
void f1(string const& texte);
{
...
}
```

Surcharge des opérateurs

Par une fonction indépendante (amie ou non) :

```
class T {
    friend T operator+(const T& t1, const T& t2);
    friend istream& operator>>(istream& in, T& t);
};
ostream& operator<<(ostream& out, const T& t);
```

Par une fonction membre :

```
class T {
public :
    T& operator=(const T& t);
    T& operator++();
    T operator++(int);
};
```

Concepts non traités

- Standard Template Library (STL) : conteneurs, itérateurs, algorithmes
- L'héritage
- La sérialisation
- Les exceptions
- Les classes template
- Les bibliothèques graphiques : Qt, GTK, wxWidgets
- ...