std::string

```cpp
#include <string>
int main() {
  std::string s1; // Empty
  std::string s2 = "today"; // Initialized
  s1 = "Hello, World. How are you" + s2; // Assigning
  if (s2 != "tonight")
    s1 += "morning?";
  printf("%s", s1.c_str()); // Back to the pure C world
}
```

Flux standards

```cpp
#include <iostream>
int main() {
  int i = 0;
  std::cout << "Enter an integer: ";
  std::cin >> i;
  std::cout << "You typed "<<i<< std::endl;
  return 0;
}
```

Flux standards et fichiers

```cpp
#include <fstream>
int main() {
  std::ifstream input("myfile.txt");
  if (not input.is_open())
    std::cerr << "Error\n";
  int c;
  double d;
  input >> c >> d;
}
```

getline sur les fichiers; Flux vers des chaînes

```cpp
#include <sstream> // std::istringstream
(autres includes)
// Chaque ligne de data.txt est de la forme 'objet poids'; on veut le poids total
int main() {
  double total = 0;
  std::ifstream input("data.txt");
  for (std::string line; std::getline(input, line); ) {
    std::string obj;    double poids;
    if (line[0] != '#') { // On passe les lignes de commentaires, qui commencent par #
      std::istringstream in(line);
      in >> obj >> poids;
      total += poids;
  } }
  std::cout << "Total: " << total;
}
```

Par valeur

```cpp
struct point {
  double x = 0, y = 0;
};
void move(struct point pt){
  pt.x += 10;
  pt.y += 10;
}
int main() {
  struct point p1;
  move(p1); // p1 INCHANGÉ
}
```

Par adresse

```cpp
struct point {
  double x = 0, y = 0;
};
void move(struct point* pt){
  pt->x += 10;
  pt->y += 10;
}
int main() {
  struct point p1;
  move(&p1);
}
```

Paramètres par référence

```cpp
struct point {
  double x = 0, y = 0;
};
void move(struct point& pt) {
  pt.x += 10;
  pt.y += 10;
}
int main() {
  struct point p1;
  move(p1);// MODIF n'est pas
}   // explicite dans le code
```

Vecteurs et boucles étendues

```cpp
#include <vector>

std::vector<int> vect {1, 3, 6};
vect[3] = 24;
vect.resize(5);
vect[4] = 2;
for (int i: vect)
  std::cout << i << '\n';
```

Algorithmes standards

```cpp
#include <algorithm>

auto pos = std::find(vect.begin(), vect.end(), 42)
if (pos == vect.end())
  std::cout << "Pas trouvé";
else
  std::cout << "Trouvé en position " << pos;
std::sort(vect.rbegin(), vect.rend());
```

Usage explicite d'itérateurs

```cpp
// Ajoute les valeurs sans tenir compte des doublons (vecteur trié)
int total = 0;
std::vector<int>::iterator pos = vect.begin();
while (pos != vect.end()) {
  total += *pos;
  int last_seen = *pos;
  pos++;
  while (pos != vect.end() && *pos == last_seen) // passe les dupplicats
    pos++;
}
```

Template de fonction

```cpp
template <typename T>
T min(T a, T b) {
  return (a < b) ? a : b;
}

```

usage de ce template

```cpp
int main() {
  int    i1 = 3, i2 = 6,       i = min<int>(i1, i2);
  double d1 = 14.2, d2 = 43.1, d = min(d1, d2);
  MyFraction f1(1,4), f2(1,2), f = min(f1,f2);
}
```

Paradigmes de programmation

La famille en Prolog

```
% Des faits, séparés par des points
child(john,sue).          child(john,sam).
child(jane,sue).          child(jane,sam).
child(sue,george).        child(sue,gina).
male(sam).    male(george).
female(sue).  female(jane).  female(june).

% Des règles, également séparées par des points
parent(Y,X)  :- child(X,Y).
father(Y,X)  :- child(X,Y), male(Y).
grand_father(X,Z) :- father(X,Y), parent(Y,Z).
opp_sex(X,Y) :- male(X), female(Y).
opp_sex(Y,X) :- male(X), female(Y).

% Des requêtes (ou objectifs), et leurs réponses
?- father(sue, george)
   true.
?- grand_father(X, george)
   X = john ; X = jane.
   % Les X vérifiant l'expression
```

Recipients en TLA$^+$

```
(***************************************)
(* Getting 4 liters from a 5l and 3l jugs    *)
(***************************************)

EXTENDS Naturals \* we use numbers

VARIABLES big,   \* amount of liters in big jug
          small  \* amount of liters in small one

TypeInvariant == /\ small \in 0..3
                 /\ big   \in 0..5

(** Initial state **)
Init == big = 0 /\ small = 0

(** Possible transitions **)
FillSmallJug  ==  small' = 3     /\ big' = big
FillBigJug    ==  small' = small /\ big' = 5
EmptySmallJug ==  small' = 0     /\ big' = big
EmptyBigJug   ==  small' = small /\ big' = 0

Min(m,n) == IF m < n THEN m ELSE n

SmallToBig == /\ big'   = Min(big + small, 5)
              /\ small' = small - (big' - big)
BigToSmall == /\ small' = Min(big + small, 3)
              /\ big'   = big - (small' - small)

(** Next-state relation **)
Next ==  \/ FillSmallJug
         \/ FillBigJug
         \/ EmptySmallJug
         \/ EmptyBigJug
         \/ SmallToBig
         \/ BigToSmall

(** Complete specification **)
Spec == Init /\ [][Next]_<<big, small>>

(* Property of which we want a counter-example *)
NotSolved == big # 4
```

A cat catches a birds and eats it

Version OOP

```
class Bird;
class Cat {
  void catch(Bird b);
  void eat();
private:
  Bird prey;
}

int main() {
  auto cat = new Cat();
  auto bird = new Bird();
  cat.catch(bird);
  cat.eat();
}
```

Version FP

```
class Bird;
class Cat;
class Catcher;
class CatReplete;

Catcher
  catch(Cat c, Bird b);
CatReplete
  eat(Catcher c);

int main() {
  std::cout <<
    eat(catch(Cat(),
              Bird()));
}
```