1 Chaînes de caractères

std::string
```
#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 old C world
}
```

2 Flux d'entrée/sortie

Flux standards
```
#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
```
#include <fstream> // file stream
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
```
#include <sstream> // string stream

// Chaque ligne de data.txt de forme 'objet poids'
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 commentaires
      std::istringstream in(line);
      in >> obj >> poids;
      total += poids;
  } }
  std::cout << "Total: " << total;
}
```

3 Paramètres de fonction

Par valeur
```
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
```
void move(struct point* pt){
  pt->x += 10;
  pt->y += 10;
}
int main() {
  struct point p1;
  move(&p1); // Modification (écriture lourde)
}
```

Par référence ; valeur par défaut
```
void move(struct point& pt, int incr = 10) {
  pt.x += incr;
  pt.y += incr;
}
int main() {
  struct point p1;
  move(p1); // Modification (pas explicite)
  move(p1, 20); // autre valeur d'incrément
}
```

4 Vecteurs

Vecteurs et boucles étendues

```
#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';
```

Usage explicite d'itérateurs

```
// Ajoute les valeurs triées en dé-dupliquant
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)
  pos++;  // passe les dupplicats
}
```

Algorithmes standards

```
#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());
```

5 Programmation générique et templates

Template de fonction

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

```

usage de ce template

```
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);
}
```