

Initiation au C++

Bernard Hugué

2 janvier 2018

Table des matières

1	Méta	1
2	Qu'est-ce que le C++ ?	1
3	Un langage polyvalent et multi-paradigme	4
4	Programmation Orientée Objet	4
5	Remarques générales	4
6	Types	4
7	Syntaxe	8
8	Passages d'arguments	11
9	Bibliothèque standard	11
10	Sémantique	14
11	Exemples	16
12	Programmes	18

1 Méta

1.1 Objectifs

Faciliter l'apprentissage ultérieur du C++

1.2 Prérequis

Comprendre le langage Python et savoir écrire des programmes en Python

1.3 Méthode

Pratique : Essayer d'exécuter les fragments de code, par exemple sur :

- <https://www.onlinegdb.com/>
- <http://coliru.stacked-crooked.com/>

2 Qu'est-ce que le C++ ?

Par rapport à Python, le C++ est un langage :

- compilé
- avec typage statique

2.1 Compilation

Le code C++ n'est pas exécuté directement et il n'y a pas de console (REPL) C++. Les programmes en C++ sont "traduits" (compilés) en *exécutables*. <https://godbolt.org/>

2.1.1 Inconvénients de la compilation

- pas d'interactivité
- les exécutables sont spécifiques à l'architecture (CPU & Système d'exploitation)

2.1.2 Avantages de la compilation

- rapidité d'exécution
- permet de tirer parti du *typage statique* pour avoir des erreurs de compilations (!)
- (permettra l'écriture de programme parallèles)

2.1.3 compilation en pratique

utilisation d'un compilateur sur chaque fichier source, puis édition de liens.

```
1 gcc fichier-source.cxx -o fichier-executable
```

2.2 Typage statique

Toutes les expressions ont un type connu à la compilation (**avant** l'exécution!)

```
1 def is_inside_unit_circle(x, y):  
2     return x*x + y*y < 1
```

```
1 bool is_inside_unit_circle(double x, double y){  
2     return x*x + y*y < 1;  
3 }
```

2.2.1 Typage générique

```
1 template<T> bool is_inside_unit_circle(T x, T y){  
2     return x*x + y*y < 1;  
3 }  
4  
5 bool test= is_inside_unit_circle(0.1, 0.1);
```

2.2.2 Typage automatique

```
1 std::string str("test");  
2 auto str2=str + " concat";  
3 int i= 5;  
4 auto j = i/2;  
5 std::cout<< str2 << " " << j;
```

test concat 2

2.2.3 conversions de type

-Wconversion -Werror

```
1 int c= 13.5f;  
2 float d= c/2;  
3 unsigned int e = -1;  
4 std::cout<< c << " " << d << " " << e <<std::endl;
```

```
error: conversion to 'int' alters 'float' constant value [-Werror=float-conversion]
int c= 13.5f;
    ~~~~

cc1plus: all warnings being treated as errors
```

```
1  int c= static_cast<int>(13.5f);
2  float d= static_cast<float>(c/2);
3  unsigned int e = static_cast<int>(-1);
4  std::cout<< c << " " << d << " " << e <<std::endl;
5  std::cout<< (-1 % 4) <<std::endl;
```

13 6 4294967295
-1

2.3 Le type peut indiquer qu'une valeur est constante

```
1  import math
2  math.pi= 2
3  print(math.cos(math.pi/2))
```

0.5403023058681398

```
1  const double pi= std::acos(-1);
2  // nice try
3  pi= 2.0;
4  std::cout << std::cos(pi/2.) << std::endl;
```

```
error: assignment of read-only variable 'pi'
pi= 2.0;
    ~~~
```

2.4 Erreurs de compilation

Détection automatique des erreurs de type.
Mieux qu'une erreur à l'exécution !

```
1  n= input("Entrez le nombre de fléchettes à lancer")
2  inside= 0
3  for i in range(n):
4      if is_inside_unit_circle(random.random(), random.random()):
5          inside +=1
6  print("Sur {} fléchettes, {} à l'intérieur. Pi ~ {}".format(n, inside, 4 * inside/n) )
```

2.5 Performance

```
1  import random
2  import time
3
4  def is_inside_unit_circle(x, y):
5      return x*x + y*y < 1
6
7  n= 100000000 #int(input("Entrez le nombre de fléchettes à lancer"))
8  inside= 0
9  start= time.time()
10 for i in range(n):
11     if is_inside_unit_circle(random.random(), random.random()):
12         inside +=1
13 print("simulation faite en {} secondes.".format(time.time() - start))
14 print("Sur {} fléchettes, {} à l'intérieur. Pi ~ {}".format(n, inside, 4 * inside/n) )
```

simulation faite en 63.49054312705994 secondes.
Sur 100000000 fléchettes, 78538271 à l'intérieur. Pi ~ 3.14153084

```

1  #include <iostream>
2  #include <random>
3  #include <chrono>
4
5
6  bool is_inside_unit_circle(double x, double y){
7      return x*x + y*y < 1;
8  }
9
10 int main(int argc, char* argv[]){
11     std::default_random_engine generator;
12     std::uniform_real_distribution<double> distribution(0.0,1.0);
13     //std::cout<<"Entrez le nombre de fléchettes à lancer :";
14     long n= 100000000; // std::cin >> n ;
15     long inside= 0;
16     auto start= std::chrono::system_clock::now();
17     for(long i=0; i != n; ++i){
18         if (is_inside_unit_circle(distribution(generator), distribution(generator))){
19             ++inside;
20         }
21     }
22     auto nanosecs= std::chrono::system_clock::now() - start;
23     std::cout<<"simulation faite en "<< nanosecs.count()/10.e9 << " secondes."<<std::endl;
24     std::cout<<"Sur "<<n<<" fléchettes, "<< inside << " à l'intérieur. Pi ~ "
25         << (4. * inside)/n << std::endl;
26 }

```

simulation faite en 0.209304 secondes.

Sur 100000000 fléchettes, 78544111 à l'intérieur. Pi ~ 3.14176

2.6 Remarques sur la performance en python

Il est possible d'écrire des programmes performants en Python !

En utilisant des bibliothèques qui ne sont pas implémentées en python (e.g. Numpy) (Elles sont souvent implémentées en C++ !)

3 Un langage polyvalent et multi-paradigme

On peut programmer à peu près n'importe quoi et n'importe comment !

Le C++ a **beaucoup** évolué depuis 20 ans ! (→ chercher des sources récentes)

4 Programmation Orientée Objet

Le C++ **permet** la Programmation Orientée Objet, mais nous ne nous y intéresserons pas (cf. cours de Java).

5 Remarques générales

Pour utiliser des bibliothèques, on utilise la directive **#include**.

Le programme est une fonction `int main(int argc, char* argv[]){}`.

6 Types

6.1 types primitifs

6.1.1 entiers

Généralement, on utilise des types qui ne sont pas précisément spécifiés pour pouvoir correspondre avec l'architecture de compilation (e.g. 32 bits vs 64 bits).

- char
- short

— int
— long
— long long
signed (défaut) ou unsigned.
std::size_t pour les indices positifs.

```
1  std::cout<<"sizeof(char): "<< 1 <<std::endl;
2  std::cout<<"sizeof(short): "<< sizeof(short) <<std::endl;
3  std::cout<<"sizeof(int): "<< sizeof(int) <<std::endl;
4  std::cout<<"sizeof(long): "<< sizeof(long) <<std::endl;
5  std::cout<<"sizeof(long long): "<< sizeof(long long) <<std::endl;
6  std::cout<<"sizeof(std::size_t): "<< sizeof(std::size_t) <<std::endl;
```

```
sizeof(char): 1
sizeof(short): 2
sizeof(int): 4
sizeof(long): 8
sizeof(long long): 8
sizeof(std::size_t): 8
```

1. Débordements

6.1.2 virgule flottante

— float
— double
Comme en python :

```
1  if(0.1 * 3 == 0.3){
2      std::cout<<"on peut rêver..." <<std::endl;
3  }else{
4      std::cout<<"... ou pas !" <<std::endl;
5  }
```

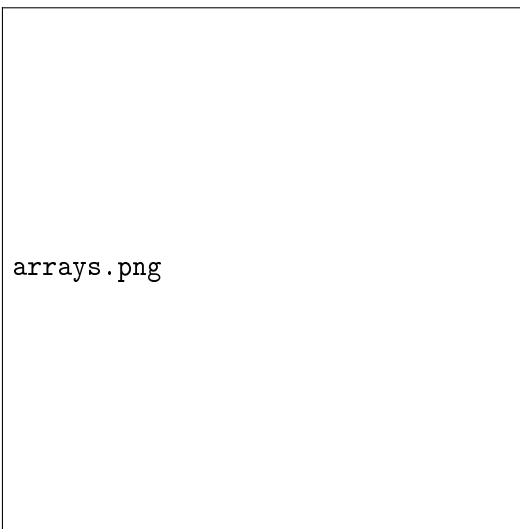
... ou pas !

6.1.3 booléen

6.2 Tableaux, pointeurs, itérateurs

6.2.1 Tableaux

valeurs de même type, contiguës en mémoire la valeur représentant le tableau est en fait son adresse en mémoire (pas d'information sur le nombre d'éléments !)



```

1  int array_i[]={1,0,-1,2};
2  double array_d[]={0.5, 6.,1.2};
3  std::cout <<'@'<<array_i<<": "<<array_i[0]<<", @"<<(array_i+1)<<": "<<array_i[1]<<std::endl;
4  std::cout <<'@'<<array_d<<": "<<array_d[0]<<", @"<<(array_d+1)<<": "<<array_d[1]<<std::endl;

```

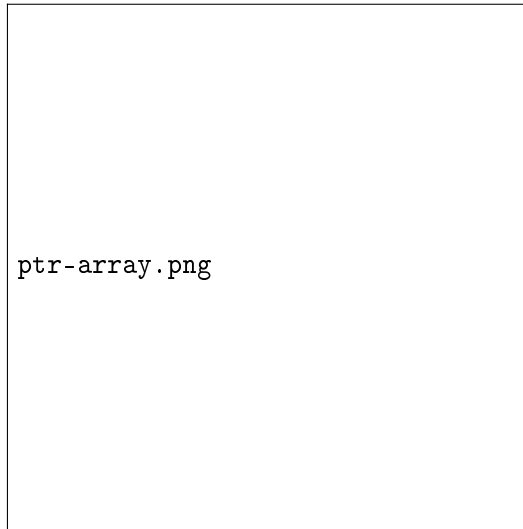
```

@0x7ffcc6f00970: 1, @0x7ffcc6f00974: 0
@0x7ffcc6f00950: 0.5, @0x7ffcc6f00958: 6

```

6.2.2 Pointeurs

adresse (typée !) en mémoire



```

1  int a=0;
2  int *ptr_a = &a;
3  *ptr_a = 2;
4  std::cout<<"a= "<< a << std::endl;
5  double arr[] = {-1.5, 0.5, 0.2};
6  double * ptr_arr= arr; // = &arr[0];
7  *ptr_arr=0.25;
8  ++ptr_arr; *ptr_arr= 1.;
9  std::cout<< "arr[0]:" << arr[0]<<", arr[1]:" << arr[1]<< std::endl;
10 std::cout<< "ptr_arr[-1]:" << ptr_arr[-1]<<", ptr_arr[0]:" << ptr_arr[0]<< std::endl;

```

```

a= 2
arr[0]:0.25, arr[1]:1
ptr_arr[-1]:0.25, ptr_arr[0]:1

```

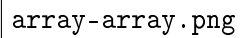
6.2.3 tableaux de tableaux

```

1  int a[4][3]={{-1, 4, -1}, {0,1,3}, {2, 1,0}, {0,0,0}};
2  std::cout<< "a[0][1]="<< a[0][1]<<" **a="<< **a << std::endl;

```

```
a[0][1]=4 **a=-1
```

array-array.png

6.2.4 Itérateurs

Généralisation du concept de pointeur pour traverser des ensembles de valeurs.

```
1  std::unordered_set<std::string> data({"red", "blue", "green"});
2  for(auto it= data.begin(); it != data.end(); ++it){
3      std::cout<< *it<<" ,";
4  }
```

green ,blue ,red ,

6.3 chaîne de caractères

pointeur sur un tableau de caractères (`char`) terminé par `'\0'`

ptr1-str.png

1. Attention aux caractères non ASCII!

```
1 char const * const str_ascii="e";
2 char const * const str_not_ascii="é";
3 std::cout << "strlen("<< str_ascii <<")="<<std::strlen(str_ascii) << std::endl;
4 std::cout << "strlen("<< str_not_ascii <<")="<<std::strlen(str_not_ascii) << std::endl;
```

strlen(e)=1

strlen(é)=2

2. Attention aux comparaisons et affectations les opérations s'appliquent aux **pointeurs** :

```
1 char * str="toto";
2 char str2[]= {'t','o','t','o', '\0'};
3 std::cout<<"str:"<<str<<" str2:"<<str2<<" (str == str2): "<< (str == str2)<< std::endl;
4 char* str3= str2;
5 str3[1]='i';
6 std::cout<<"str2:"<<str2<<std::endl;
```

str:toto str2:toto (str == str2): 0

str2:tito

6.4 std::string

std::string : *classe* permettant de faire de opération sur des chaîne de caractères (par exemple les copier et les comparer!)

Nécessite un `#include <string>`.

```
1 #include <iostream>
2 #include <string>
3 int main(int argc, char* argv[]){
4     std::string str="toto";
5     char tmp[]= {'t','o','t','o', '\0'};
6     std::string str2=tmp ;
7     std::cout<<"str:"<<str<<" str2:"<<str2<<" (str == str2): "<< (str == str2)<< std::endl;
8     std::string str3= str2;
9     str3[1]='i';
10    std::cout<<"str2:"<<str2<<std::endl;
11 }
```

str:toto str2:toto (str == str2): 1

str2:toto

7 Syntaxe

7.1 instructions

séparées par des ;

7.2 blocs de code

Délimités par des { et } et non pas indiqués par l'indentation (qui reste utilisées, mais seulement pour permettre la visualisation).

```
1 while x % 2 == 0:
2     if x > 0:
3         print("strictement positif")
4     elif x < 0:
5         print("strictement négatif")
6     else:
7         print("nul")
```

```
1 while(x % 2 == 0){
2     if (x > 0){
3         std::cout<<"strictement positif"<<std::endl;
4     }else if (x < 0){
5         std::cout<<"strictement positif"<<std::endl;
6     }else{
7         std::cout<<"nul"<<std::endl;
8     }
9 }
```

7.3 ATTENTION aux blocs "manquants" !

Les blocs sont syntaxiquement facultatifs.

Une instruction vide ; est valide.

```
1 int x= -1;
2 if( x > 0)
3     std::cout << "positif" << std::endl;
4     std::cout << "still positif ?" << std::endl;
5 std::cout << "not indented" << std::endl;
```

still positif ?

not indented

```
1 int x= -1;
2 if( x > 0);
3     std::cout << "positif" << std::endl;
4     std::cout << "still positif ?" << std::endl;
5 std::cout << "not indented" << std::endl;
```

positif

still positif ?

not indented

7.4 redéfinition d'opérateurs

Le sens de certain opérateurs dépend de ce quoi à ils s'appliquent ! «, *,...

```
1 int a=1;
2 a= a << 1;
3 int *ptr_a=&a;
4 a= a * *ptr_a;
5 std::cout << "a="<< a << std::endl;
```

a=4

7.5 Exécution conditionnelle : if

L'expression testée est fausse si elle vaut 0, vraie sinon.

```
1  int i=55;
2
3  if(i){
4      std::cout<< i << " est vrai" <<std::endl;
5  }else{
6      std::cout<< i << " est faux" <<std::endl;
7  }
8  std::cout<< "true vaut "<< true << std::endl;
```

55 est vrai
true vaut 1

7.6 Évaluation conditionnelle : ?:

l'expression (e ? t : v) est équivalente à l'expression python `scr_python[:export code]{t if e else v}`

```
1  int i=55;
2
3  std::cout<< i << " est "<< ( i ? "vrai" : "faux") <<std::endl;
4  std::cout<< "true vaut "<< true << std::endl;
```

55 est vrai
true vaut 1

7.7 Exécution conditionnelle : switch

branchement conditionnel selon différentes valeurs entières pour une expression testée. Attention au break!

```
1  int i= 10;
2  switch (i){
3  case 0: { std::cout<< " i vaut zéro"; break;}
4  case 10: {std::cout <<" i vaut dix"; }
5  case 20: {std::cout <<" i vaut vingt (ou j'ai oublié le break avant)"; break;}
6  default:{ std::cout << "i vaut "<<i;}
7  }
```

i vaut dix i vaut vingt (ou j'ai oublié le break avant)

7.8 Boucle while

Comme la boucle `while` en python, mais il existe aussi une variante `do{}while()`;

```
1  int next;
2  do{
3      next= std::rand() % 100;
4  }while((next % 2 == 0) || (next % 3 == 0));
5  std::cout<<next<<" n 'est divisible ni par 2 ni par 3";
```

83 n 'est divisible ni par 2 ni par 3

7.9 Boucle for

```
for( initialisation; test; mise à jour)
```

```
1 for(std::size_t i=0; i != 5; ++i){
2     std::cout<<i<<' ';
3 }
4 std::cout<<std::endl;
5 char* str="toto";
6 for(int i=0; str[i] != '\0'; ++i){
7     std::cout<<str[i]<<' ';
8 }
9 std::cout<<std::endl;
10 for(char* ptr=str; *ptr; ++ptr){
11     std::cout<< *ptr << '_';
12 }
13 std::cout<<std::endl;
```

```
0 1 2 3 4
t o t o
t_o_t_o_
```

7.10 Boucle for sur une séquence

Comme en python mais avec typage :

```
1 int array[]={1, 5, 7, -3};
2 for(int v : array){
3     std::cout<< v <<',';
4 }
```

```
1,5,7,-3,
```

8 Passages d'arguments

Les arguments sont passés par valeur sauf si l'on indique un passage par référence.

```
1 void par_valeur(int i){
2     i+= 1;
3 }
4 void par_reference(int& i){
5     i+= 1;
6 }
7
8 int main(int argc, char* argv[]){
9     int j= 1;
10    par_valeur(j);
11    std::cout<< "j= "<< j << std::endl;
12    par_reference(j);
13    std::cout<< "j= "<< j << std::endl;
14 }
```

```
j= 1
j= 2
```

9 Bibliothèque standard

9.1 Structures de données

9.1.1 tuple

L'utilisation de la classe `std::tuple` nécessite `#include <tuple>`

```

1  std::tuple<int, float> t_if(2, 0.5f);
2  std::tuple<int, std::string> t_is= std::make_tuple(0, "toto");
3
4  std::get<0>(t_is)= std::get<0>(t_if);
5  std::cout<< std::get<0>(t_is)<<' '<<std::get<1>(t_is)<<std::endl;

```

2,toto

9.1.2 array

L'utilisation de la classe `std::array` nécessite `#include <array>`. La taille fait partie du type (donc fixe). Joue pour les tableaux le même rôle que `std::string` pour les chaînes de caractères.

```

1  std::array<int, 3> a{-1, 2, 0};
2  auto b= a;
3  std::array<int, 3> c{-1, 2, 0};
4  b[0]= 5;
5  std::cout << a[0] << ", (a == c): " <<(a== c)<<std::endl;

```

-1, (a == c): 1

9.1.3 vector

`std::vector` est semblable aux listes de python, mais avec éléments de même type. Nécessite `#include <vector>`

```

1  std::vector<int> v_i;
2  v_i.push_back(1);
3  v_i.push_back(2);
4  v_i.push_back(4);
5  v_i.pop_back();
6  std::cout<<v_i.size()<<" elts:"<<v_i[0]<<","<<v_i[1]<<std::endl;

```

2 elts:1,2

9.1.4 list

`std::list` permet des opérations efficaces en tête de liste (*front*). Ne permet pas un accès aléatoire. Nécessite `#include <list>`

```

1  std::list<int> c;
2  c.push_front(1);
3  c.pop_back();
4  c.push_front(2);
5  c.push_front(4);
6  std::cout<<"c.empty():"<<c.empty()<<" , elts:";
7  auto it=c.begin();
8  std::cout<< *it<<' ';
9  ++it;
10 std::cout<<*it<<std::endl;

```

c.empty():0, elts:4,2

9.1.5 unordered_map

`std::unordered_map` est équivalent aux dictionnaires de python, mais avec des types.

Nécessite `#include <unordered_map>`.

```

1  std::unordered_map<std::string, int> name_to_score;
2  name_to_score["toto"]=5;
3  name_to_score["bernard"]= 32;
4  name_to_score["patrick"]= 64;
5  for(auto const & kv : name_to_score){
6      std::cout<<kv.first<<" : "<<kv.second<<std::endl;
7  }

```

```
patrick : 64
toto : 5
bernard : 32
```

9.1.6 unordered_set

`std::unordered_set` permet de tester **efficacement** si un ensemble contient un élément, et d'éviter les doublons.

```
1  std::unordered_set<std::string> names;
2  names.insert("toto");
3  names.insert("titi");
4  names.insert("toto");
5  std::string to_find("titi");
6  if(names.find(to_find) != names.end()){
7      std::cout << to_find << " trouvé dans ";
8  }
9  for(auto name : names){
10     std::cout<< name << ", ";
11 }
```

titi trouvé dans titi, toto,

9.2 Itérateurs

Les structures de données, entre autres!, donnent accès aux éléments à travers des *itérateurs*. Ils sont catégorisés selon les opérations qu'ils permettent (en plus de l'accès par l'opérateur `*` et de la comparaison par l'opérateur `==`).

9.2.1 forward iterator

Incrémentation pour passer à l'élément suivant avec l'opérateur `++`.

```
1  std::ostream_iterator<int> out(std::cout, ", ");
2  for(int i=0; i != 5; ++i, ++out){
3      *out= i;
4  }
```

0, 1, 2, 3, 4,

9.2.2 Bidirectional iterator

En plus des opérations du *forward iterator*, on peut aussi décrémenter pour aller à l'élément précédent :

```
1  std::list<int> c{2, 3, 5, 7};
2  auto it=c.begin();
3  std::cout<< *it << ',';
4  ++it;
5  std::cout<< *it <<std::endl;
6  --it;
7  std::cout<< *it <<std::endl;
```

2,3
2

9.2.3 Random access iterator

En plus des opérations du *bidirection iterator*, on peut aller directement à n'importe quel élément situé à `n` positions dans un sens ou dans l'autre (comme par opérations arithmétiques sur des pointeurs) :

```
1  std::vector<int> v_i {2, 3, 5, 7, 11};
2  auto it= v_i.begin();
3  it += 2;
4  std::cout<< *it <<' ' << *(it-2) <<' ' << *(it + 2) <<std::endl;
```

5 2 11

9.3 Algorithmes

On peut appliquer des algorithmes sur n'importe quel intervalle de n'importe quelle structure de données grâce à des intervalles `[begin,end]` et des itérateurs en écriture. Ils nécessitent l'inclusion du header correspondant avec `#include <algorithm>`

9.3.1 copy

`std::copy`

```
1  std::vector<int> v{2, 3, 5, 7, 11};
2  std::copy(v.begin(), v.end(), std::ostream_iterator<int>(std::cout, " "));
```

2, 3, 5, 7, 11,

9.3.2 shuffle

`std::shuffle` nécessite un générateur de nombres aléatoires (Random Numbers Generator), avec `#include <random>`. L'initialisation de celui-ci peut changer en se basant sur l'instant au moment d'exécution (cf. `#include <chrono>`).

```
1  std::vector<int> v{2, 3, 5, 7, 11};
2  std::size_t seed = std::chrono::system_clock::now().time_since_epoch().count();
3  std::shuffle(v.begin(), v.end(), std::default_random_engine(seed));
4  std::copy(v.begin(), v.end(), std::ostream_iterator<int>(std::cout, " "));
```

9.3.3 sort

On peut trier un intervalle d'éléments d'une séquence avec `std::sort`. La relation d'ordre est paramétrable.

```
1  std::vector<int> v{7, 3, 2, 5, 11};
2  std::size_t seed = std::chrono::system_clock::now().time_since_epoch().count();
3  std::shuffle(v.begin(), v.end(), std::default_random_engine(seed));
4  std::sort(v.begin(), v.end());
5  std::copy(v.begin(), v.end(), std::ostream_iterator<int>(std::cout, " "));
```

10 Sémantique

Par défaut les valeurs sont passées par **copie**.

10.1 passage par copie

```
1 void f(int i){
2     i+=1;
3 }
4 void f(std::vector<int> v){
5     v.push_back(0);
6 }
7 int main(int argc, char* argv[]){
8     int i=0;
9     f(i);
10    std::cout<< i <<std::endl;
11    std::vector<int> v{1, 2};
12    f(v);
13    std::cout<< v.size() << std::endl;
14 }
```

0

2

10.2 passage par pointeur

Attention, si le pointeur est invalide (*NULL* ou *nullptr*), toute utilisation (sauf test de comparaison avec *nullptr*) provoquera un plantage !

```
1 void f(int* i){
2     *i +=1;
3 }
4 void f(std::vector<int>* v){
5     (*v).push_back(0);
6     v->push_back(0);
7 }
8 int main(int argc, char* argv[]){
9     int i=0;
10    f(&i);
11    std::cout<< i <<std::endl;
12    std::vector<int> v{1, 2};
13    f(&v);
14    std::cout<< v.size() << std::endl;
15 }
```

1

4

10.3 passage par référence

On peut vouloir :

- éviter de payer les coûts d'une copie
 - assurer qu'il y a bien une valeur (pas un pointeur invalide)
 - permettre la modification d'une variable passée en argument
- passage par référence

```
1 void f(int& i){
2     i +=1;
3 }
4 void f(std::vector<int>& v){
5     v.push_back(0);
6 }
7 int main(int argc, char* argv[]){
8     int i=0;
9     f(i);
10    std::cout<< i <<std::endl;
11    std::vector<int> v{1, 2};
12    f(v);
13    std::cout<< v.size() << std::endl;
14 }
```

1

3

11 Exemples

Traductions en C++ de fonctions vues en DS.

11.1 Limites

```
1  std::tuple<int> limites(std::vector<int> const& xs){
2      int min_x= xs[0];
3      int max_x= xs[0];
4      for(int x : xs){
5          if(x < min_x){
6              min_x= x;
7          }
8          if( x > max_x){
9              max_x= x;
10         }
11     }
12     return std::make_tuple(min_x, max_x);
13 }
14 template<typename T>
15 std::tuple<T> limites_generic(std::vector<T> const& xs){
16     T min_x= xs[0];
17     T max_x= xs[0];
18     for(T x : xs){
19         if(x < min_x){
20             min_x= x;
21         }
22         if( x > max_x){
23             max_x= x;
24         }
25     }
26     return std::make_tuple(min_x, max_x);
27 }
```

```
1  template<typename It>
2  std::tuple<decltype(*It), decltype(*It)>
3  limites_iter(It begin, It end){
4      typedef decltype(*It) T;
5      T min_x= *begin; //!!!
6      T max_x= *begin;
7      for(It it=begin; it != end; ++it){
8          if(*it < min_x){
9              min_x= *it;
10         }
11         if( *it > max_x){
12             max_x= *it;
13         }
14     }
15     return std::make_tuple(min_x, max_x);
16 }
```

```
1  template<typename It>
2  std::tuple<decltype(*It), decltype(*It)>
3  limites_iter2(It begin, It end){
4      It min_it= begin; //!!!
5      It max_it= begin;
6      for(It it=begin; it != end; ++it){
7          if(*it < *min_it){
8              min_it= it;
9          }
10         if( *it > *max_it){
11             max_it= it;
12         }
13     }
14     return std::make_tuple(*min_it, *max_it);// !!!
15 }
16 template<typename It>
17 std::tuple<It, It> limites_idiomatic(It begin, It end){
18     It min_it= begin; //!!!
19     It max_it= begin;
20     for(It it=begin; it != end; ++it){
21         if(*it < *min_it){
```



```

22     min_it= it;
23 }
24 if( *it > *max_it){
25     max_it= it;
26 }
27 }
28 return std::make_tuple(min_it, max_it);
29 }
30
31 // En fait,  std::minmax_element() existe !

```

11.2 Fusion

```

1 // char != code point !
2 std::string fusion(std::string const& str1, std::string const& str2){
3     std::string res;
4     if(str1.size() == str2.size()){
5         for(auto it1= str1.begin(), it2= str2.begin(); it1 != str1.end(); ++it1, ++it2){
6             res.push_back(*it1);
7             res.push_back(*it2);
8         }
9     }
10    return res;
11 }
12 // char != code point !
13 template<typename It1, typename It2, typename Out>
14 Out fusion_idiomatic(It1 begin1, It1 end1, It2 begin2, It2 end2, Out out){
15     if(std::distance(begin1, end1) == std::distance(begin2, end2)){
16         for(; begin1 != end1; ++begin1, ++begin2){
17             *out= *begin1;
18             ++out;
19             *out= *begin2;
20             ++out;
21         }
22     }
23     return out;
24 }

```

11.3 Palindrome

```

1 bool palindrome(std::string const& str){
2     for(std::size_t i= 0; i != str.size()/2; ++i){
3         if(str[i] != str[str.size()-i-1]){
4             return false;
5         }
6     }
7     return true;
8 }
9 bool palindrome2(std::string const& str){
10    auto it= str.begin();
11    auto r_it= str.rbegin();
12    for(std::size_t i= 0; i != str.size()/2; ++i, ++it, ++r_it){
13        if(*it != *r_it){
14            return false;
15        }
16    }
17    return true;
18 }
19 bool palindrom_idiomatic(It begin, It end){
20    std::size_t const half= std::distance(begin, end)/2;
21    std::reverse_iterator<It> rbegin(end);
22    for(std::size_t i= 0; i != half; ++i, ++begin, ++rbegin){
23        if(*begin != *rbegin){return false;}
24    }
25    return true;
26 }

```

11.4 Extraction de données

```

1 typedef std::tuple<std::string, std::string, double> data_t;
2

```

```

3  std::vector<data_t> extraction(std::vector<data_t> const& xs, std::string crit){
4      std::vector<data_t> res;
5      for(data_t x : xs){
6          if(std::get<0>(x) == crit){
7              res.push_back(x);
8          }
9      }
10     return res;
11 }
12
13 template<typename It, typename Out>
14 Out extraction_idiomatic(It begin, It end, Out out, std::string crit){
15     for(; begin != end; ++begin){
16         if(std::get<0>(*begin) == crit){
17             *out= *begin;
18             ++out;
19         }
20     }
21     return out;
22 }
23
24 /*
25 std::copy_if(data.begin(), data.end(), std::back_inserter(selection)
26             , [category](data_t const& x)->bool{return std::get<0>(x) == category;})
27 */
28

```

```

1  typedef tuple<std::string, std::string, double> data_t;
2
3  std::vector<data_t> extract_dict(std::unordered_map<std::string, std::vector<data_t> dict, std::string cat){
4      return dict[cat];
5  }

```

11.5 Pliage

```

1  void pliage(std::string str, std::size_t n){
2      for(std::size_t i(0); i != str.size(); ++i){
3          if( (i != 0) && (i % n == 0)){
4              std::cout << std::endl;
5          }
6          std::cout<<str[i];
7      }
8  }
9
10 template<typename In, typename Out>
11 Out pliage_idiomatic(In begin, In end, Out out, std::size_t n){
12     for(std::size_t i(0); begin != end; ++begin){
13         if((i != 0) && (i % n == 0)){
14             *out= '\n';
15             ++out;
16         }
17         *out= *begin;
18         ++out;
19     }
20     return out;
21 }

```

12 Programmes

12.1 Pendu

On eut essayer de réécrire le programme de jeu de pendu en C++. Cependant, la notion de lettre n'étant pas gérée (ne pas confondre lettres et caractères!), le programme ne marchera pas avec des lettres codées sur plusieurs octets (accentuées par exemple). Pour ce genre de programmes sans impératifs de performance, python est donc plus adapté.

```

1  #include <iostream>
2  #include <string>

```

```

3  #include <vector>
4
5  std::string initialiser_mot_mystere(std::string const& mot){
6      std::string res;
7      for(std::size_t i=0; i != mot.size(); ++i){
8          res.push_back('_');
9      }
10     return res;
11 }

```

```

1  void affichage_mot_mystere(std::string const& mot_mystere){
2      for(char const& c : mot_mystere){
3          std::cout << c << ' ';
4      }
5      std::cout<<std::endl;
6  }
7
8  void affichage_coups_restants(int nb_coups_restants){
9      std::cout << "Il vous reste " << nb_coups_restants << " coups."<<std::endl;
10 }
11
12 char saisie_joueur2(){
13     std::string saisie;
14     while( saisie.size() == 0){
15         std::cout << "Lettre à tester :"<<std::endl;
16         std::cin >> saisie;
17         std::cout << std::endl;
18     }
19     return saisie[0];
20 }

```

```

1  std::string mise_a_jour_mot_mystere(char lettre, std::string const& mot, std::string const& mot_mystere){
2      std::string res;
3      for(std::size_t i=0; i != mot.size(); ++i){
4          res.push_back(mot[i] == lettre ? lettre : mot_mystere[i]);
5      }
6      return res;
7  }
8
9  std::string choix_joueur2(std::string const& mot, std::string const& mot_mystere
10                          , int nb_coups_restants){
11     affichage_mot_mystere(mot_mystere);
12     affichage_coups_restants(nb_coups_restants);
13     char lettre_saisie= saisie_joueur2();
14     return mise_a_jour_mot_mystere(lettre_saisie, mot, mot_mystere);
15 }

```

```

1  bool test_jeu_fini(std::string const& mot_mystere, int nb_coups_restants){
2      if(mot_mystere.find('_') == std::string::npos){
3          std::cout<< "Félicitations! Vous avez trouvé le mot mystère."<< std::endl;
4          return true;
5      }else if(nb_coups_restants == 0){
6          std::cout<< "Félicitations! Vous avez trouvé le mot mystère."<< std::endl;
7          return true;
8      }
9      return false;
10 }

```

```

1  int main(int argc, char* argv[]){
2      while(true){
3          std::cout << "*****" << std::endl
4              << "Bienvenu sur le jeu du Pendu" << std::endl
5              << "1 - Commencer une nouvelle partie" << std::endl
6              << "0 - Quitter" << std::endl
7              << "*****"<< std::endl;
8
9          int choix;
10         std::cout<<"Faites votre choix"<<std::endl;
11         std::cin >> choix;
12         if( choix == 0){
13             break;
14         }
15     }

```

```

14         std::cerr<<" choix:" << choix <<std::endl;
15         if( choix != 1){
16             continue;
17         }
18         std::string mot;
19         std::getline(std::cin, mot); // flush line break
20         while( mot.size() ==0){
21             std::cout << "Joueur 1 - Saisissez le mot mystère:" << std::endl;
22             std::getline(std::cin, mot);
23         }
24
25         std::string mot_mystere= initialiser_mot_mystere(mot);
26         for(std::size_t i=0; i != 100; ++i){
27             std::cout << std::endl;
28         }
29         for(int nb_coups_restants= 10; !test_jeu_fini(mot_mystere, nb_coups_restants); --nb_coups_restants){
30             mot_mystere= choix_joueur2(mot, mot_mystere, nb_coups_restants);
31         }
32     }
33 }

```

12.2 Chaos Game

Soit le programme suivant en python, qui implémente un dessin paramétrable suivant le principe du chaos game. On peut lancer ce programme de la façon suivante :

```

1 python ./chaos-game.py 4 >chaos-game-py-frame-4.pgm

```

et obtenir le fichier image **chaos-game-py-frame-4.pgm** au format PGM.

Avec le program **convert** d'ImageMagick, on peut convertir à la volée au format PNG :

```

1 python ./chaos-game.py 5 | convert pgm:- chaos-game-py-frame-5.pgm

```

pour d'obtenir l'image suivante :

chaos-game-py-frame-5.png

On voudrait faire varier progressivement le paramètre de 3.00 à 8.00 avec un incrément de 0.01 pour générer une animation. Comme le programme python met 2 minutes par image, les 500 images nécessaires prendraient 1000 minutes, soit beaucoup trop longtemps ! On va donc réécrire le programme en C++.

12.2.1 Programme en Python

Implémentation en python

```
1  import sys
2  import math
3  import random
4
5
6  def midpoint(p0, p1):
7      (x0, y0)= p0
8      (x1, y1)= p1
9      return ((x0 + x1)/2, (y0 + y1)/2)
10
11 def rotate(center, a, p):
12     (x0, y0)= center
13     (x, y)= p
14     x-= x0
15     y-= y0
16     return (x0+math.cos(a)*x - math.sin(a)*y,
17             y0+math.sin(a)*x + math.cos(a)*y)
18
```

```

19 def polygon(center, p, n):
20     res=[]
21     for i in range(math.ceil(n)):
22         res.append(p)
23         p= rotate(center, 2*math.pi/n, p)
24     return res
25
26 def brighten(screen, p, white):
27     (x,y)= p
28     x= int(x)
29     y= int(y)
30     screen[y][x]= min(screen[y][x]+1, white)
31
32 def create_screen(init, w, h):
33     return [[init for x in range(w)] for y in range(h)]
34
35 def print_image(white, screen):
36     print("P2")
37     print(len(screen[0]), len(screen[1]), sep=" ")
38     print(white)
39     for line in screen:
40         for c in line :
41             print(c, end=" ")
42     print()
43
44 def main(argv):
45     n_edges= float(argv[0]) if len(argv)>0 else 3.
46     w= 1024
47     h= 1024
48     black= 0
49     white= 255
50     n= int(n_edges * ( int(argv[1] if len(argv) > 1 else 1000000)))
51     poly= polygon((w/2, h/2), (w/2, 1.75*h/2), n_edges)
52     screen= create_screen(black, w, h)
53     prev_idx= 0
54     p= poly[prev_idx]
55     for i in range(n):
56         idx= random.randrange(len(poly))
57         while idx == prev_idx:
58             idx= random.randrange(len(poly))
59         prev_idx= idx
60         p= midpoint(poly[idx], p)
61         brighten(screen, p, white)
62     print_image(white, screen)
63 if __name__ == "__main__":
64     main(sys.argv[1:])

```

12.2.2 Réécriture partielle en C++

Au lieu de 17 heures pour générer les 500 images, on peut réduire le temps d'exécution à 17 minutes sur la base du programme suivant.

```

1  #include <iostream>
2  #include <tuple>
3  #include <cmath>
4  #include <cstdlib>
5  #include <vector>
6  /*
7  STEPS=100;
8  PREFIX=chaos-game-frame-${STEPS}
9  for i in $(seq $(3 * $STEPS) $(8 * $STEPS)); do
10     ./chaos-game-double $(bc -l <<< "scale=2;$i / $STEPS") | convert pgm:- ${PREFIX}-${i}.png;
11 done
12 ffmpeg -start_number 300 -i ${PREFIX}-%03d.png -filter_complex "[0:v]reverse,fifo[r];[0:v][r] concat=n=2:v=1 [v]" -map "[v]" \
13 chaos-game.mp4
14 */
15
16 typedef std::tuple<double, double> point_t;
17
18 point_t midpoint(point_t const& p0, point_t const& p1){
19     // TODO
20 }
21
22 point_t rotate( point_t const& center, float a, point_t const& p){

```

```

23     double const x= std::get<0>(p)-std::get<0>(center);
24     double const y= std::get<1>(p)-std::get<1>(center);
25
26     return point_t(std::get<0>(center)+std::cos(a)*x - std::sin(a)*y
27                   , std::get<1>(center)+std::sin(a)*x + std::cos(a)*y);
28 }
29
30 const double pi= std::atan(1)*4;
31
32 std::vector<point_t> polygon(point_t const& center, point_t p, float n){
33     std::vector<point_t> res;
34     for(int i= 0; i < std::ceil(n); ++i){
35         res.push_back(p);
36         p= rotate(center, 2*pi/n, p);
37     }
38     return res;
39 }
40 // TODO implementer brighten
41
42 std::vector<std::vector<int> > create_screen(int init, std::size_t w, std::size_t h){
43     std::vector<std::vector<int> > screen(h, std::vector<int>(w, init));
44     return screen;
45 }
46
47 template<typename C> std::ostream& operator<<(std::ostream& os, std::vector<std::vector<C> > const& screen){
48     for(auto row : screen){
49         for(auto c : row){
50             os << c << ' ';
51         }
52     }
53     return os;
54 }
55
56 int main(int argc, char* argv[]){
57     float n_edges= argc > 1 ? std::atof(argv[1]) : 3.f;
58     int n_e = static_cast<int>(n_edges);
59     int const w= 1024;
60     int const h= 1024;
61     int const white= 255;
62     int const black= 0;
63     std::size_t const n= (argc > 2 ? std::atol(argv[2]) : 10000000)*n_edges;
64
65     auto poly= polygon(point_t(w/2, h/2), point_t(w/2, 1.75*h/2), n_edges);
66     auto sc= create_screen(black, w, h);
67     std::size_t prev_idx= 0;
68     point_t p= poly[0];
69     for(std::size_t i=0; i != n; ++i){
70         std::size_t idx=0;
71         do{
72             idx= std::rand()%poly.size();
73         }while(idx == prev_idx);
74         p= midpoint(poly[idx], p);
75         prev_idx= idx;
76         brighten(sc, p, white);
77     }
78     std::cout << "P2" << std::endl << w << ' ' << h << std::endl << white << std::endl;
79     std::cout<< sc << std::endl;
80     return 0;
81 }

```
