Chapitre 2 : variables, entrée/sortie dans un fichier, nombres aléatoires

Fondamentaux du C++

Master 2 Ingénierie mathématique
Sorbonne Université

Automne 2018

Plan

Organisation de la mémoire

- Mémoire de l'ordinateur : long "ruban" de cases mémoire= suites de 8 bits (=1 *octet*).

Déclaration de variable : réservation d'un certain nombre de cases mémoire (place nécessaire pour stocker la variable).
Exemples :

- `bool` = 1 bit = 1 case mémoire ;
- `char` = 8 bits = 1 case mémoire ;
- `int` = 32 bits = 4 cases mémoire ;
- `double` = 64 bits = 8 cases mémoire.

- adresse mémoire d'une variable `x` accessible par `&x` ;
- `&x` est un objet de type *pointeur* ;
- `std::cout << &x` : affiche l'adresse de la variable.

Visibilité des variables

Visibilité d'une variable : le fait qu'une variable ne puisse pas être utilisée à n'importe quel moment. La visibilité dépend des conditions dans laquelle la variable a été déclarée.

Trois types de variables :

Variables locales

- Une variable définie à l'intérieur d'un bloc n'est visible que dans ce bloc ;
- *Bloc* = groupe d'instructions situé entre deux accolades ;
- Utilisations typiques : variable interne à une fonction, paramètre d'une boucle.

Visibilité des variables

Visibilité d'une variable : le fait qu'une variable ne puisse pas être utilisée à n'importe quel moment. La visibilité dépend des conditions dans laquelle la variable a été déclarée.

Trois types de variables :

Arguments de fonction

- Accessible seulement à l'intérieur de la fonction correspondante ;
- Chaque appel de la fonction définit une nouvelle variable dans laquelle on recopie la valeur passée en argument de la fonction ;
- Notamment, une fonction ne peut pas modifier ses arguments ;
- On verra plus tard qu'une modification des arguments nécessitera de passer par des pointeurs ou des références.

Visibilité des variables

Visibilité d'une variable : le fait qu'une variable ne puisse pas être utilisée à n'importe quel moment. La visibilité dépend des conditions dans laquelle la variable a été déclarée.

Trois types de variables :

Variables globales

- Déclarées hors de toute fonction ;
- Accessibles à tout moment ;
- Attention : ce type de variable peut être modifié sans que l'on s'en aperçoive (par un appel de fonction).
- De préférence, déclarer les variables globales comme étant constantes.
- Usage typique : paramètres d'un problème à ne pas modifier.

Visibilité des variables

- Dans un même programme, plusieurs variables peuvent avoir le même nom.
- La "plus locale" des deux va masquer la "plus globale".

Trop multiplier les variables différentes ayant le même nom est une mauvaise idée pour avoir un code lisible.

Qualificateur `const`

Une variable peut être déclarée *constante* par la syntaxe

```
type const nom;
```

Cela provoque une erreur à la compilation si la variable est modifiée dans le programme : c'est une aide pour l'écriture du programme.

Utilisation

On conseille de déclarer constante toutes les variables dont on sait qu'elles ne doivent pas être modifiée, afin de détecter toute modification accidentelle.

Qualificateur `static`

Une variable locale peut être déclarée *statique*, par la syntaxe

```
static type nom;
```

Donne une durée de vie permanente à la variable, sans changer sa visiblilité : la variable ne va pas être réinitialisée à chaque passage dans le bloc où elle est déclarée.

Exemples

Compter le nombre d'appel à une fonction ; garder la valeur courante d'un générateur de nombres aléatoires.

Plan

Écriture et lecture dans un fichier

- bibliothèque `fstream`.
- deux *types* `std::ofstream` (sortie (out) vers un fichier) et `std::ifstream` (entrée (in) depuis un fichier).

```
std::ofstream mon_flux; //On déclare un flux de sortie
mon_flux.open("mon_fichier"); // On ouvre un fichier
mon_flux << "Bonjour\n"; // On écrit dans le fichier
mon_flux.close(); // On referme le flux
```

On peut aussi ouvrir le fichier dès la déclaration :

```
std::ofstream mon_flux("mon_fichier");
```

Refermer le fichier n'est pas indispensable, mais c'est un bon réflexe, par exemple si on manipule de nombreux fichiers.

Écriture et lecture dans un fichier

On verra plus tard que `std::ofstream` et `std::ifstream` sont en fait des *classes*, que

```
std::ofstream mon_flux("mon_fichier")
```

est un appel de constructeur, et que

```
mon_flux.open("fichier")
```

et

```
mon_flux.close()
```

sont des appels de méthodes.

Plan

Nombres aléatoires : la théorie

- Ordinateur : aléa impossible.
- *Mais* : il existe des algorithmes *totalement déterministes*, générant des suites vérifiant certaines propriétés statistiques.
- On parle de suites de nombres *pseudo-aléatoires*.

En pratique

- On calcule $x_n \in \{1, \ldots, N\}$, avec N grand, "ressemblant" à une suite i.i.d. de loi uniforme sur $\{1, \ldots, N\}$.
- Si X est uniforme sur $\{1, \ldots, N\}$, alors X/N est approximativement uniforme sur $[0, 1]$.

Un (*mauvais*) exemple : RANDU

On définit $x_n \in \{1, \ldots, 2^{31} - 1\}$ par :

$$\begin{cases} x_0 & \text{donné,} \\ x_{n+1} & = (2^{16} + 3) \times x_n \text{ modulo } 2^{31}. \end{cases}$$

$(x_n)_{n \in \mathbb{N}}$ est $2^{29} \simeq 5 \times 10^8$-périodique. Pour être un "bon" générateur, un générateur doit (entre autres) avoir une longue période. Ici, la période 2^{29} peut être suffisante pour bon nombre de problèmes.

Un (*mauvais*) exemple : RANDU

- Autre condition : corrélations faibles
- Ici : pour tout n $x_{n+2} = 6x_{n+1} - 9x_n \mod 2^{31}$. La suite des $(x_n, x_{n+1}, x_{n+2})/N$ n'est pas du tout uniformément répartie dans $[0,1]^3$.

Remarque

La suite $(x_n)_{n\in\mathbb{N}}$ est indexée par le paramètre x_0, appelé *graine* du générateur.
On obtient plusieurs suites différentes en partant de plusieurs graines différentes.
Deux suites partant de deux graines différentes *ne doivent par contre pas être considérées commme des suites indépendantes*.

Les implémentations : `rand`

La méthode simple

- Fonction `rand` de la bibliothèque `cstdlib`.
- Insuffisant pour certaines applications (méthode de Monte Carlo nécéssitant une longue période).
- Dans certaines implémentations, `rand` a une période de $2^{15} \simeq 32.000$: *beaucoup trop peu*.
- Choix de la graine avec `srand`. Le résultat de `rand` est un entier compris 0 et `RAND_MAX` inclus.

Les implémentations : Mersenne twister

La méthode efficace

- Depuis le standard 2011 (avec g++, option "`-std=c++11`"), le générateur dit "Mersenne twister" est implémenté ;
- Très utilisé pour les calculs de Monte Carlo (période extrêmement grande : $2^{19937} - 1 \simeq 4.32 \times 10^{6001}$) ;
- Pas sûr pour un usage cryptographique.

Les implémentations : Mersenne twister

Remarque : $2^{19937} - 1$ vaut

```
4315424797388162648055235516337919839053935043226711505165250541403330680137658091130451362931858466554526993825764883531790221733458441390952826915460916801900
7875343741396296801920114486480902661414318443276980300066728104984095451588176077132969843762134621790396391341285205627619600513106646376648615994236675486537
4802419643502959351686623639090479483476923139783013778207857124190544743328445291831729732423108882650813216264694510777078122828294447750226804880578200287646
5939916476626520090056149580034405435369038986289406179287201112083361480844748291354732836727787956564830784690911694586623016970240126024018702874665003344577
4570315431292996025187780790119375902863171084149642473378986267503308961374905766340905289572290016038000571630875191373979555047468154333253474991046248132504
5163417965514705754814592008594726148362138755571168644457897508862779964873043084504842234206292665185560243393391908443689210184248446770427276646018529149252
7728092269753842677025733392895440120546589561034765885538663390254628996213264328242574803578623358060815469654693256383332767076989943977488852668727852745100
2963059146963875715425735534475979734463100678367393327402149930968778296741391514599602374213629898720611431410402147238998090962818915890645693934483330994169
6322958779958489933667470148717634948055499961630515412254034652970077211462313557040814930986630657336771911728539870957481678162560842128233801686253345864312
5403467080613527354327071447887686186198332077728064480669112571319726258176315131359642954776357636783701934983517846214429496075719091805462511414366638418943
3852576452289347652454631535740468786228945885654608562058042468987372436921445092315377698407168198376538237748614196207041548106379365123192817999006621766467
1671134716327154817958770053826943934004030617004576911353491878748889234293493401451705717161811257958888892774954269771499145496239163940148229850253316515114
3127880200905680845650681887726660983163688388490562182226293398654864566908067219170474040889134983568566242806323119852043682632941529075297279834342944650999
2206368781367154091702655772727391329424277529349082600585884766523150957417077831910016168475685658673192860882070179760307269849987354836042371734660257694347
2355063017441188741412924389581415491006097522168822308876114319964723308423801371109274494835578150375868496445857499177728699267442183696211376751010832785437
9408174909409104308409677414470843632427947689205620042722796163866914980548983112124467639993195537148401288636074870647956866904857478285521705474011394592962
2177502575565811067452201448981991968635965361551681273982740760138899638820318776303668762730157584640042798880691862640268612686180883874939573818125022279689
9302674462557739595424698316378630001712792271514060341299021815706596505326007758236773981821290873944498591827499990072235924233345678506711865688391867477049
6001627754062533144061901912998378991471251536520033605799350860167880768756856237785709525554130490292719222018417250235712444991187021064269456506138491937347
4324503966267799038402386781686809962015879090586549423504699190743519551043722544515740967829084336025938225780730880273855261551972044075620326780624448803490
9982321612316877947156134057932495455095280525180101230872587789741158170482455889714385967544080813134383755029887267395233752966416155014060916079832292398272
4061478325289247971651993698951918780868122119164174771090248063349109170482744122828118663244590714578713835123484226138007462191400481815238666604313334487506
7903582838283562688083236575482068479639546383819532174522502682372441363275765875609119783653298312066708217149316773564340379289724393986744139891855416612295
7393566686126582712346964383771228389980401997390780614436754156710784634046737024037776534781733670848447347020568666361581380036922533822099094664695919301616
2609792050874217567030650513954286075080615983535754103214709508427846105670136773979493202420299870773101769258204621070221251412042932253043178961626704777611
5123597935404147084870985465426502772057300900333847905334250604119503030001704002887892941404603345869926367501355094942750552591581639980523190679610784993580
8966832992976812624423140086570334218680945517405064488290392073167113076951318922965935090186230948105575195603052407871638092191644337545148633010009159169858
5624217656362477132898167854824629737624953025136036341276836645617507703197745753491280643317653999599434330811847014715871281614939442127661422826290995005574
6981053206610001560295784656616193252269412026831159508949671513845195883217147982748879261851417819979034417285598607727220866677680426090308754823803345446566
3056192413083744527546681430154877108777280110860043258922622594139682852834970455710627577014217615652627251534074076254051499319894944591064146605343053785767
0986252004986488096114486925860347371436365919401396270636685138929969286949180517255681850829882495495481579606316951765874142015979875427342802672345248126356
9157307213153739781041627653715078598504154797287663122946711348158529418816432825044466692781137474494898385064375787507376496345148625306383391555145690087891
9553159944629444932352488175999071191357559333821217061914771850549366322111572229203311485024875633031180188056850735698415805181187107786539535712960143729408
6527040702192438316729032323156791228941948624059403907445232167801938187121909215546076844457357855951361330424220615135645751393727093900970723782710124585383
7678338161023397586854894230696091540249987907453461311923963852950754758058205625956600817743007191746812655955021747670922460866747744520875607859062334750627
0983285934800677894561696024943928137634956575998474857735539909575573132008090408300364464922194099340969487305474943012161656867507357495558823403039898746729
7545506095773692155919548081551403591570712993005702711728625284319741331230761788679750678426019543676030599034070848146460727895549548774214075357062121719825
2192978869786916734625618430175454903864111585429504569920905636741539030968041471
```

Les implémentations : Mersenne twister

La méthode efficace

- Type `std::mt19937`, défini dans la bibliothèque `random` ;
- `std::mt19937 gen(1234)` déclare une variable `gen` correspondant à un générateur de graine 1234 ;
- Suite pseudo-aléatoire obtenue par des appels successifs de `gen()` ;
- La valeur maximale rendue est `gen.max()` ;
- On verra plus tard que `std::mt19937` est en fait une classe, dont `std::mt19937 x(int)` est un constructeur, et que `x()` correspond à l'appel de l'opérateur `()`.

Les implémentations : Mersenne twister

La méthode efficace

- On peut déclarer des variables correspondant à des lois de probabilité. Par exemple `std::normal_distribution<double> G(0,1);`.
- `G` va alors correspondre à la loi Gaussienne de moyenne 0 et de variance 1 .
- Chaque appel de `G(gen)` renvoie une réalisation de cette variable.