curvilignes "{...}" pour délimiter le bloc d'instructions devant êtres répétées (étapes 2 et 3 dans l'algorithme ci-dessus), et à l'intérieur de ce bloc, un second bloc imbriqué également délimité par des "{...}" indiquant l'instruction sujette à une exécution conditionnelle (étape 3 de l'algorithme). Notez également que la fonction `rand` est une fonction prédéfinie du langage C, équivalente à un dé à 2147483648 faces, dont chaque appel produit un entier aléatoire extrait d'une distribution uniforme dans l'intervalle $[0, 2147483647]$; cette dernière valeur n'est en fait applicable qu'aux architectures montées en 64 bits, comme pour la quasi-totalité des ordinateurs que vous serez susceptible d'utiliser. Il existe toute une batterie de ces fonctions prédéfinies, correspondant aux fonctions mathématiques avec lesquelles vous êtes déjà familiers, comme `sin`, `cos`, `exp`, `log`, etc.

Le résultat de la compilation du code source produit un fichier dit **exécutable**. Ce fichier est typiquement illisible même aux programmeurs d'expérience. Le détail de l'exécutable dépend également du compilateur utilisé pour la traduction du code source en exécutable. Sur la plupart des systèmes LINUX, la compilation du code source se ferait comme suit. On suppose ici que le code source est contenu dans un fichier texte nommé `estimePi.c`. Il suffit alors de taper:

```
gcc estimePi.c
```

Ceci créera dans le répertoire courant un fichier appelé `a.out`, qui, sur la plupart des système Linux, est par la suite exécuté en tapant simplement

```
./a.out
```

Pour le code source ci-dessus roulant sur mon vieux Mac, l'instruction `printf` produit alors la sortie suivante:

```
valeur de pi = 3.142452
```

ce qui approxime bien π (= 3.1415926536...) à 0.04% près. Comme vous pouvez certainement l'imaginer, un algorithme donné peut s'exprimer de différentes manières même si écrit dans le même langage de programmation. C'est comme au primaire, quand on vous demandait une production écrite de 10 lignes sur *Le Brontosaure*; votre texte pouvait bien dire essentiellement la même chose que celui d'un(e) de vos petit(e) camarade, mais son style était assurémement différent, et certains styles se lisent mieux que d'autres. Il y a également des styles en programmation, qui souvent visent un compromis entre la lisibilité du code source, la modularité, et la vitesse d'exécution du code une fois compilé. Par exemple, le code suivant produit exactement la même sortie que le précédent, aussi rapidement, mais pour un(e) débutant(e) est un peu plus opaque:

```
#include <stdio.h>
int main(void)
{
  long i, x, y, n=10000000, s=0 ;
  for (i=1; i<=n ; i++) {
    x=rand() ; y=rand() ;
    if (x*x+y*y <= 2147483647*2147483647) s += 1 ;
  }
  printf ("valeur de pi = %f\n",4.*s/n) ;
}
```

(toujours pour un ordi monté en 64 bits). Personnellement, c'est plutôt comme ça que j'aurais tendance à coder l'algorithme (quoique j'y ajouterais certainement quelques explications en commentaires). Comme en littérature ou en musique, le goût personnel y est également pour quelque chose, mais certains aspects du style de programmation sont généralement considérés incontournables. Revenant à notre première version du code C, notez par exemple que: