« Petits » programmes en C

Gilles Menez - UNSA - UFR Sciences - Dépt. Informatique

5 septembre 2017

Objectifs pédagogiques

Il s'agit de vous familiariser avec :

- ➢ la structure d'un programme C,
- ➢ les variables,
- ➢ les types de base,
- ➢ les expressions C,
- ➢ les instructions, ...

Table des matières

Enoncé Conditionnel : if

Cette section a pour but de vous montrer un exemple de la structure de contrôle conditionnelle fournie par l'instruction `if` .

1 Nombre caché (20 minutes)

```c
/* Fichier : Codes/devine1.c */
#include <stdio.h>
#define SUP 100
int main (int argc, char*argv[]){
  int x;
  x = rand();

  printf("x = %d\n",x);
  return 0;
}
```

Sémantique des lignes de code :

① Prologue,

② Déclaration des variables/fonctions relatives au E/S

③ Définition d'une constante : **SUP**

④ Définition du point/fonction d'entrée du pgm : `main` ,

⑤ Définition de la variable x de type **int**

⑥ Appel de la fonction `rand`

⑦ Affectation du résultat dans x

⑧ Appel de la fonction d'affichage de haut niveau : `printf`

⑨ Terminaison du programme

Sur ce premier exemple on utilise la fonction `rand()` pour tirer un nombre au hasard :

(a) Compiler et exécuter deux ou trois fois.
Vous devriez constater que la mise en oeuvre de l'aléatoire produit des résultats pour l'instant très déterministes.

(b) Modifier le code pour obtenir une valeur différente de x à chaque exécution : Il faut lire le manuel de la fonction `rand` ... notion de "seed" !

(c) Modifier le code pour limiter l'intervalle de valeurs du nombre "caché" à $[0, SUP]$ où SUP est une constante macro-définie.

Une piste de solution : l'opérateur modulo.

(d) Modifier le code pour limiter l'intervalle de valeurs du nombre caché à $[INF, SUP]$ où INF et SUP sont deux constantes macro-définies.

Il n'y a pas d'opérateur particulier/remarquable qui pourrait aider. Il faut enchaîner correctement les opérations.

(e) Souvent vous avez besoin de demander une donnée au clavier pour que le programme s'exécute.
Analyser, comprendre la solution proposée dans ce qui suit, puis intégrer la gestion du INF et le SUP dans votre programme.

```c
/* Fichier : Codes/devine2.c

Histoire d'utiliser le if, ce programme :
  a) choisit un nombre caché entier
  b) demande à l'utilisateur de deviner ce nombre
  c) affiche le résultat
 */
#include <stdio.h>
int main (int argc, char*argv[]){
  int nb_cache;     /* Choisi aleatoirement  */
  int proposition; /* Faite par l'utilisateur */
  int nblus; /* Nombre de tokens lus */

  nb_cache = rand();
  /*printf("nb_cache  = %d\n",nb_cache);*/

  printf("Proposez une valeur pour le nombre cache : ");
  nblus = scanf("%d",&proposition);
  if (nblus == 1){
    printf("Votre proposition est : %d\n",proposition);
    if ( proposition != nb_cache)
      printf("Mauvaise proposition !\n");
    else
      printf("Bonne proposition !\n");
  }
  else
    printf("Bizarre comme proposition ?\n");
  return 0;
}
```

(f) Refaire avec Python à partir de l'exemple suivant :

```python
'''
Created on 27 juin 2016
@author: GM
'''
if __name__ == '__main__':
    SUP = 100
    INF = 10
    input_value = raw_input("Quelle est votre proposition ? ")
    try:
        # try and convert the string input to a number
        val = int(input_value)
        if val >= 0 :
            print("plus de blabla !")
        else:
            print("moins de blabla !")
    except ValueError:
        # tell the user off
        print "Aie !! {input} is not a number, \
        please enter a number only".format(input=input_value)
```

Enoncé `if` en Python : `https://docs.python.org/3.4/reference/compound_stmts.html#if`

2 Résolution des équations du second degré (30 minutes)

Pour continuer à pratiquer les énoncés conditionnels :

(a) Faire un programme (dans le langage de votre choix C ou Python) de résolution des équations du second degré de la forme

$$a * x^2 + b * x + c = 0$$

avec $a, b, c \in \Re$.

(b) Réaliser une présentation *élégante* des solutions (indiquer le nombre de racines réelles distinctes, nombre de solutions complexes,...).

Pour autant vous n'utiliserez pas la notion de complexe disponible en C et en Python.

(c) Proposer des jeux de valeurs permettant de tester les différents cas de figure.

(d) Tester votre programme avec le jeu de coefficients suivants : $a = 0.25$, $b = 0.1$ et $c = 0.01$

Vous aurez remarqué que c'est un cas de figure où le discriminant est "théoriquement" nul.

Enoncés Itératifs

3 Calcul de π (60 minutes)

On veut disposer d'un programme écrit en langage C capable d'effectuer le calcul de π.

➢ Plusieurs algorithmes sont envisageables.

D'un point de vue "informatique", il s'agit de manipuler les variables de cumul, les premières fonctions et les itérations. Sur le plan algorithmique, il faut maîtriser les convergences des algorithmes proposés.

➢ C'est une application directe de ce que l'on a fait en cours !

3.1 Première méthode :

Dans un premier temps, on utilisera la formule de Leibniz issue du développement en série de Taylor de `arctan(x)` pour $x \in [-1, 1]$:

$$\pi/4 = \arctan(1) = \sum_{n=0}^{n=\infty} (-1)^n/(2n+1) \tag{1}$$

(a) Dans un fichier **meth1.c**, situé dans le répertoire adéquat, réaliser la fonction `arct_m1` calculant ainsi π.

Ce fichier ne contient qu'une fonction, le point d'entrée du programme (la fonction main) se trouve dans une autre fichier.

Il est important de comprendre qu'un programme **ce n'est pas qu'un seul fichier**.

(b) Dans un fichier **main.c**, réaliser le programme (fonction main) permettant d'invoquer cette fonction.

(c) Compiler.

La compilation d'un programme C distribué sur plusieurs fichiers n'a rien de difficile. Vous "donnez" au compilateur, tous les fichiers qu'il doit intégrer dans la fabrication du fichier exécutable (ici `leibniz`) :

gcc main.c meth1.c -o leibniz -lm

(d) Tester.

Pour information et pour vérification, une estimation très précise de π (obtenue *via* le logiciel `MAPLE`)

```
$ maple
$ evalf(Pi,1000) ;
```

est :

```
3.14159265358979323846264338327950288419716939937510582097494459230781640628620899862803482534211706798214808651328230664709384460955058223172535940812848111745028410270193852110555964462294895493038196....
```

En C, la constante `M_PI` (<math.h>) permet de disposer d'une approximation (à la précision de la machine) de π.

```
#define M_PI           3.14159265358979323846  /* pi */
```

Il peut être intéressant de faire afficher la différence entre l'estimation obtenue par notre calcul et la valeur "référence".

Concrètement : Il faut remplir les fichiers suivants !

```c
/*-------------------------------------------------------------
   Fichier : meth1.c

   Calcul de Pi : Formule de Leibniz
 -------------------------------------------------------------*/

#include <stdio.h>
double arct_m1(){
  /*
     Calcul de pi par la formule de Leibniz
     Il manque sans doute choses et notamment des paramètres ?
  */


  /* !!!!!!!!!! Code de la fonction !!!!!!!!! */
}

/*-------------------------------------------------------------*/

/*-------------------------------------------------------------
   Fichier : main.c

   Test des fonctions/algorithme de calcul de pi
 -------------------------------------------------------------*/

#include <stdio.h>

double arct_m1();
int main(int argc, char*argv[]){
  double res= 0.0;

  /* Appel de la fonction arct_m1() */
  res = arct_m1();
  printf("Le resultat de la methode de Leibniz est : %f\n",res);
}
/*-------------------------------------------------------------*/
```

3.2 Deuxième méthode :

On peut aussi utiliser la formule de John Machin pour calculer π :

$$\pi/4 = 4 * \arctan(1/5) - \arctan(1/239) \tag{2}$$

avec :

$$\arctan(x) = \sum_{n=0}^{n=\infty} (-1)^n * \frac{x^{2n+1}}{2n+1} \tag{3}$$

(a) Dans un fichier **meth2.c**, créer une fonction de calcul de l'arctangente `my_arct`.

(b) Dans le même fichier, créer `arct_m2` qui est censé correspondre à cette deuxième formule et qui utilise la fonction `my_arct`.

Attention à l'écriture des choses en informatique :

➢ Quelle différence entre 1/5 et 1.0/5.0 ?

(c) Invoquer cette fonction depuis la fonction principale (`main`) située dans le fichier **main.c** déjà créé pour de la première méthode.

Vous avez désormais un programme formé de trois fichiers.

(d) Compiler et tester.

TODO plus tard ?

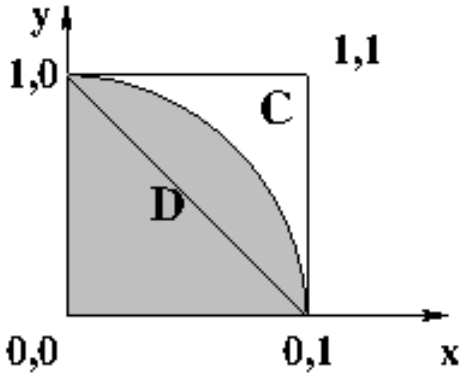
① Avez vous des remarques/apports à faire sur la méthode/algo ?
cf `http://www.pi314.net/fr/machin.php`

② Un truc à regarder : la convergence si on se contente de tester un epsilon . . . et qu'on l'a choisi trop petit.

3.3 Troisième méthode :

Le principe de cette méthode consiste à tirer aléatoirement un grand nombre de points à l'intérieur du carré C délimité par les points $(0, 0)$ et $(1, 1)$.

➡ La proportion des points intérieurs au quart de disque D de rayon 1 et de centre $(0, 0)$ tend vers le rapport des surfaces de D sur C, c'est-à-dire $\pi/4$.



Le programme effectue donc une boucle qui tire un point au hasard et incrémente un compteur si le point se trouve être intérieur à D ($x^2 + y^2 \leq 1$).

(a) Dans un fichier **meth3.c**, proposer une fonction réalisant cette (très mauvaise) estimation de la valeur de π.

Pour générer un nombre aléatoire compris entre 0 et 1, cette fonction s'appuiera sur la fonction donnée ci-dessous :

```
/*-------------------------------------------------------------------------
double uniform() - generates zero mean uniform random number from -0.5 to 0.5
Returns one zero mean uniformly distributed random number as a double.
No arguments, calls function rand() from the C library.
-------------------------------------------------------------------------*/
/* RAND_MAX define gives the biggest value rand() will return and is used only
if not already defined (non-ANSI implementations of C may require this) */
#ifndef RAND_MAX
#define RAND_MAX 32767
#endif

double uniform(){
  return((double)(rand() & RAND_MAX) / RAND_MAX - 0.5);
}
/*------------------------------------------------------------------------- */
```

(b) Invoquer cette fonction depuis la fonction principale (`main`).

(c) Compiler et tester.

TODO :

① Avez vous des remarques/apports à faire sur la méthode/algo ?

3.4 Bilan :

On aimerait bien savoir quelle méthode est la meilleure ?

- ✔ Qu'est ce que vous proposez comme critère ?
- ✔ Pour un n donné, l'écart à la valeur vraie sur 15 décimales ?
- ✔ Pourquoi pas une courbe qui représenterait cet écart versus n ?
  Pour les trois méthodes ?

  Voir page 14 du document/cours sur matplotlib !

3.5 Recherche itérative (30 minutes)

On dépasse les 2 heures du TD, on commence cette semaine et on finira la prochaine fois !

```c
#include <stdio.h>

int main (int argc, char*argv[]){
  int nb_cache;     /* Choisi aleatoirement  */
  int proposition; /* Faite par l'utilisateur */

  nb_cache = rand(); /* à borner par  INF et SUP ! */
  /*printf("nb_cache  = %d\n",nb_cache);*/

  while (1){
    printf("Proposez une valeur pour le nombre cache : ");
    scanf("%d",&proposition);
    printf("Votre proposition est : %d\n",proposition);

    if ( proposition != nb_cache)
      printf("Mauvaise proposition !\n");
    else{
      printf("Bonne proposition !\n");
      break;
    }
  }
  return 0;
}
```

Cette évolution de l'exercice du début permet d'itérer sur la proposition d'un nombre caché permettant ainsi à l'utilisateur de faire plusieurs propositions et de mettre en place un algorithme de recherche . . . un peu efficace ?

(a) Proposer une évolution de l'instruction `while` permettant de se passer de l'instruction de rupture de séquence : `break` .

(b) Proposer une évolution permettant de fournir des informations (facilitant la recherche) à l'utilisateur :

"trop grand" / "trop petit"

(c) Remplacer l'utilisateur "humain" par des lignes de codes.
Ici, il s'agit de traduire le raisonnement qui vous permet de trouver le nombre en un algorithme puis un programme de recherche.

- ➟ Première solution, "on" part de l'INF (première proposition) et on incrémente jusqu'au succès.
- ➟ Deuxième solution, plus efficace !
  C'est la notion de recherche par dichotomie. Cela commence ainsi :
  1. Si INF et SUP bornent l'espace des solutions entières possibles, je propose le milieu.
  2. Si ce milieu est trop grand, l'espace des solutions peut être réduit, et milieu devient SUP.
  3. Si ce milieu est trop petit, la solution se trouve dans $[milieu, SUP]$. INF est affecté avec milieu.
  4. Et on recommence, . . . on choisit un nouveau milieu . . .