Review of Scientific Programming in C and Fortran

Michael McLennan
Software Architect
HUBzero™ Platform for Scientific Collaboration
Monte Carlo Simulator

Simulate by randomly generating thousands of tracks.

50/50 chance left or right at each peg.

Count

Bucket
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#define LEVELS 9

int main(int argc, char **argv) {
    int max, drop, i, pos, count[LEVELS+1];
    double rnum;

    printf("number of drops?\n");
    if (scanf("%d", &max) != 1) {
        fprintf(stderr,"bad number!\n");
        exit(1);
    }

    for (i=0; i < LEVELS+1; i++) {
        count[i] = 0;
    }

    Definitions of functions we’ll use below
    Constant value, substituted wherever it is referenced in the rest of the file

    for (i=0; i < LEVELS+1; i++) {
        count[i] = 0;
    }
}
Main program must be defined like this:

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#define LEVELS 9

int main(int argc, char **argv)
{
    int max, drop, i, pos, count[LEVELS+1];
    double rnum;

    printf("number of drops?\n");
    if (scanf("%d", &max) != 1) {
        fprintf(stderr,"bad number!\n");
        exit(1);
    }
    for (i = 0; i < LEVELS + 1; i++) {
        count[i] = 0;
    }
    for (i = 0; i <= LEVELS + 1; i++) {
        count[i] = 0;
    }
}
```

Number of drops?
500

“%d” → max

Quit the program and indicate that it failed:
exit(0) → “ok”
exit(1) → “failure”
```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#define LEVELS 9

int main(int argc, char **argv) {
    int max, drop, i, pos, count[LEVELS+1];
    double rnum;
    printf("number of drops?\n");
    if (scanf("%d", &max) != 1) {
        fprintf(stderr,"bad number!\n");
        exit(1);
    }
    for (i=0; i < LEVELS+1; i++) {
        count[i] = 0;
    }
    for (i=0; i < LEVELS+1; i++) {
        count[i] = 0;
        if (scanf("%d", &max) != 1) {
            fprintf(stderr,"bad number!\n");
            exit(1);
        }
    }
}
```

Done once before loop
Determines when to quit
Increment i at bottom of loop

- count[0] = 0;
- count[1] = 0;
- count[2] = 0;
- ...

C arrays start at 0
for (drop=0; drop < max; drop++) {
    pos = LEVELS;
    for (i=0; i < LEVELS; i++) {
        pos += (drand48() < 0.5) ? -1 : 1;
    }
    count[pos/2]++;
}
/* print out final results */
printf("Statistics:\n");
for (i=0; i < LEVELS+1; i++) {
    printf("Bucket %d: %d\n", i, count[i]);
}
return 0;

Check random number
Less than 0.5, go left
Otherwise, go right
Add on to value:
pos += 1;
pos = pos + 1;
Lather, rinse, repeat...
```c
... for (drop=0; drop < max; drop++) {
    pos = LEVELS;
    for (i=0; i < LEVELS; i++) {
        pos += (drand48() < 0.5) ? -1 : 1;
    }
    count[pos/2]++;
}

/* print out final results */
printf("Statistics:\n");
for (i=0; i < LEVELS+1; i++) {
    printf("Bucket %d: %d\n", i, count[i]);
}
return 0;

Statistics:
Bucket 0: 2
Bucket 1: 7
Bucket 2: 23
...
“Bucket %d: %d”
i  count[i]

Same as exit(0)
Everything is “ok”
```
Compiling and Running C Code

$ gcc -g plinko.c -o plinko -lm

$ ./plinko

number of drops? 500

Statistics:
Bucket 0: 1
Bucket 1: 14
Bucket 2: 24
Bucket 3: 87
Bucket 4: 137
Bucket 5: 102
Bucket 6: 95
Bucket 7: 29
Bucket 8: 10
Bucket 9: 1

Add debugging info for later
Create executable called “plinko”
Include math library for drand48()
C Language Cheat Sheet

Conditionals:
```c
if (x > 0) {
    statements;
}

if (x > 0) {
    statements;
} else if (x < 0) {
    statements;
} else {
    statements;
}

switch (x) {
    case 1:
        statements;
        break;
    case 2:
    default:
        statements;
}
```

Looping:
```c
while (x != 0) {
    statements;
}

do {
    statements;
} while (x < 10);

for (x=0; x < 10; x++) {
    statements;
}

break    ←  Break out of loop
continue →  Go back to top of loop
```
program plinko

implicit none
integer levels
parameter ( levels=9 )

integer max, drop, i, pos,
+ count(levels+1)
double precision rnum

write(6,*) 'Number of drops?'
read(5,*) max

c     set all counts to zero
    do 10 i=1,levels+1
    count(i) = 0;
    continue
10   continue

**Plinko Simulator in Fortran**

```fortran
program plinko

implicit none
integer levels
parameter ( levels=9 )

integer max, drop, i, pos,
+        count(levels+1)
double precision rnum

write(6,*) 'Number of drops?'
read(5,*) max

c    set all counts to zero
    do 10 i=1,levels+1
        count(i) = 0
    continue

10 continue
```

- **Fortran assumes...**
  - i-n → integer
  - a-h o-z → real
  - This turns that off

- Defines a constant

- **Fortran arrays start at 1**
do 20 drop=1,max
  pos = levels
  do 30 i=1,levels
    if (rand().lt.0.5) then
      pos = pos - 1
    else
      pos = pos + 1
    endif
  30     continue
  count(pos/2+1) = count(pos/2+1) + 1
20   continue

conditional operators:
  .lt.  less than
  .le.  less than or equal to
  .gt.  greater than
  .ge.  greater than or equal to
  .eq.  equal to
  .ne.  not equal to
  .and. logical and
  .or.  logical or

don’t care about the format

write out final results
write(6,*) 'Statistics:'
do 40 i=1,levels+1
  write(6,99) i, count(i)
40   continue
99   format('Bucket ',i5,' : ',i5)
end
Fortran Cheat Sheet

Conditionals:

```fortran
if (x .gt. 0) then
  statements
endif

if (x .gt. 0) then
  statements
elseif (x .lt. 0) then
  statements
else
  statements
endif
```

Looping:

```
"while loop"
10  if (x .lt. 10) then
    statements;
    goto 10
  endif

"do-while loop"
20  continue
    statements;
    if (x .lt. 10) goto 20

"for loop"
30  do 30 x=1,10,2
    statements;
  continue
```
Compiling and Running Fortran Code

$ g77 -g plinko.f -o plinko

Add debugging info for later

Create executable called “plinko”

$ ./plinko

Number of drops?

500

Statistics:

Bucket 1: 1
Bucket 2: 9
Bucket 3: 28
Bucket 4: 77
Bucket 5: 141
Bucket 6: 124
Bucket 7: 75
Bucket 8: 35
Bucket 9: 8
Bucket 10: 2
Follows the instructions in a “make” file

<table>
<thead>
<tr>
<th>File: Makefile</th>
</tr>
</thead>
<tbody>
<tr>
<td>plinko: plinko.c</td>
</tr>
<tr>
<td>TAB</td>
</tr>
<tr>
<td>gcc -g plinko.c -o plinko -lm</td>
</tr>
<tr>
<td>clean:</td>
</tr>
<tr>
<td>TAB</td>
</tr>
<tr>
<td>rm -f *.o plinko</td>
</tr>
</tbody>
</table>

No changes, does nothing

Clean up and start from scratch
What if something goes horribly wrong?

```
$ ./plinko
   Number of drops?
500
Segmentation Fault (Core Dumped)

$ gdb plinko
(gdb) l
4
5     #define LEVELS 9
6
7     int
8     main(int argc, char **argv)
9     {
10        int max, drop, i, pos, count[LEVELS+1];
11        double rnum
12
13        printf("number of drops?\n");
(gdb) break 13
Breakpoint 1 at 0x80484c5: file plinko.c, line 13.
```
(gdb) break 13
Breakpoint 1 at 0x80484c5: file plinko.c, line 13.
(gdb) run
Starting program: /home/nanohub/mc/bootcamp2008/plinko/c/plinko

Breakpoint 1, main () at plinko.c:13
13      printf("number of drops?\n");

(gdb) n
number of drops?
14      if (scanf("%d", &max) != 1) {

(gdb) n
500
19      for (i =0; i < LEVELS+1; i++) {

(gdb) n
20      count[i] = 0;

(gdb) n
19      for (i =0; i < LEVELS+1; i++) {

(gdb) p i
$1 = 1
(gdb) break 24 if drop == 3
Breakpoint 2 at 0x8048540: file plinko.c, line 24.

(gdb) c
Continuing.

Breakpoint 2, main () at plinko.c:24
24    pos = LEVELS;

(gdb) p drop
$2 = 3

Cheat Sheet

l line ... list source code (starting at optional line)
break line ... stop at this line
run arg arg ... run program with these arguments
n ... next
s ... step (step inside routines)
c ... continue running to next breakpoint
p expr ... print out value of expression