CS29003 Algorithms Laboratory

Assignment No: 1

Last date of submission: 16–January–2018

A black box has four secret integers a_1, a_2, a_3, a_4 in the range $[1, N]$, where $N = 10^8$ for this assignment. You play four games with the black box to identify the secret integers. The rules of the four games are different from one another. In each game, you are allowed to make integer-valued **guessing queries** to the black box which, in turn, returns appropriate values. When you think that you have guessed the secret correctly, validate your result by making a **verification query** to the black box.

Game 1: In this game, you make a linear search for a_1, that is, make guessing queries in the sequence $a = 1, 2, 3, \ldots$. The black box returns 0 if $a \neq a_1$ or 1 if $a = a_1$. So you stop when the black box returns 1. Write a function `playgame1()` to implement this game.

Game 2: In this game, you keep on making random guessing queries a in the range $[1, N]$. Like in Game 1, the black box returns 0 if $a \neq a_2$ or 1 if $a = a_2$. So you stop when you receive the response 1 from the black box. Write a function `playgame2()` to implement this game. You do not have to make any effort to avoid repeated random queries on the same guesses.

A random guess a is equal to the secret a_2 with probability $1/N$. Therefore after making $\Theta(N)$ queries, you expect to identify the secret a_2. In practice, your guesses may be so unlucky that the number of iterations becomes $\omega(N)$. The probability of such an event is however very low. An algorithm of this type is called a *randomized algorithm* of the *Las Vegas* type. Such an algorithm has some expected runtime guarantees, and whenever it terminates, it outputs the correct result.

In Games 1 and 2, there is no limit on how many guessing queries you may make to the black box. In the remaining two games, you are allowed to make only a limited number of queries. The black box will supply you more information than YES/NO, which you should use as hints to quickly arrive at the secret.

Game 3: In this game, the maximum number of queries allowed is about $2\lceil \lg N \rceil$. For $N = 10^8$, $\lceil \lg N \rceil = 27$, and the black box actually allows up to 56 queries. In this case, the black box returns one of the four integers $0, 1, 2, 3$. Suppose that your current query is a, and your previous query was p. Then, the return values should be interpreted in the following manner. If you make an error (like you did not register or did not initialize the game), the return value is 0. If $a = a_3$, the return value is 1. If your current guess is better than or equally distant from the previous guess, that is, if $|a - a_3| \leqslant |p - a_3|$, the return value is 2. Finally, if your current guess is poorer than the previous guess, that is, if $|a - a_3| > |p - a_3|$, the return value is 3. During your *first* guessing query, there is no previous query, so you disregard the return value (unless it is 1).

Write a function `playgame3()` to implement a binary-search-like procedure. Let $[L, R]$ be the interval where the secret is known to exist. Initially, $L = 1$ and $R = N$. You make two guessing queries on L and R. If any such query results in a return value of 1, you are done. Otherwise, you gain the information which of L and R is closer to the secret. Halve the search interval accordingly. Stop when the interval size reduces to one (that is, when you have $L = R$).

Game 4: This game is identical to Game 3 with the exception that now you are allowed to make about $\lceil \lg N \rceil$ queries (for $N = 10^8$, the actual limit is 28). The return values $0, 1, 2, 3$ are to be interpreted exactly as in Game 3 (with a_3 replaced by a_4). Write a function `playgame4()` to identify a_4.

Handling the black box

The black box is presented to you as a precompiled binary file `blackbox1.o`. Depending upon your compiler (`gcc` or `g++`), download the appropriate file, and store it in the directory where you are working.

Insert the following lines globally (that is, outside all functions) near the beginning of your program (that is, before you actually call them). These functions are implemented in the code of the black box. The compiler links the black box after your program gets compiled. These directives keep the compiler happy by informing it the exact prototypes of the functions.

```
extern void registerme ( );
extern void startgame ( int );
extern int guess ( int );
extern void verifysoln ( int );
```

At the beginning of your **`main()`** function, call **`registerme()`** to initiate the black box. You cannot do anything without registering yourself. You start each game by calling **`startgame(n)`**, where $n \in \{1,2,3,4\}$ is the game number. In order to make a guessing query on a, call **`guess(a)`**. The return value is to be interpreted as detailed in the descriptions of the games. The black box entertains guessing queries outside the interval $[1,N]$, but why do you need to make any such query? When you think that you have guessed a secret a correctly, call **`verifysoln(a)`** in order to validate your hunch. Also, record and print the running times of the four games. For example, you should play Game 1 as follows.

```
#include <time.h>

clock_t c1, c2;
int a;

registerme();

startgame(1);
c1 = clock(); a = playgame1(); c2 = clock();
printf("\n+++ Game 1\n    a1 = %d\n", a);
printf("    Time taken = %lf sec\n", (double)(c2-c1)/(double)CLOCKS_PER_SEC);
verifysoln(a);
```

Notice that **`verifysoln`** ends the running game. You cannot make any further guessing queries before starting (or restarting) another (or the same) game. If you exceed the query limit in Games 3 and 4, the blackbox terminates your program.

Depending upon your compiler, compile your program as follows.

```
gcc myprog.c blackbox1.o
g++ myprog.cpp blackbox1.o
```

Sample output

```
+++ Game 1
    a1 = 64965365
    Time taken = 0.524319 sec
*** Yep! Your guess is correct.

+++ Game 2
    a2 = 78161829
    Time taken = 1.295109 sec
*** Yep! Your guess is correct.

+++ Game 3
    a3 = 4633204
    Time taken = 0.000002 sec
*** Yep! Your guess is correct.

+++ Game 4
    a4 = 81685977
    Time taken = 0.000001 sec
*** Yep! Your guess is correct.
```

Submit a single C/C++ source file. Do not use global/static variables.