

Lesson 33

Objectives: -

- Write down program to calculate option price
- Write down program of tower of hanoi

Option Price Calculator

Function

This program calculates option prices.

Programming Issues

Usage

Type:

```
option -h
```

for the syntax.

option.c

```
/*
```

```
OPTION.C  
=====
```

Simple command line option price calculator.
Uses the Black-Scholes
formula to price european puts and calls on
non-dividend paying
stocks.

(c) Paul Griffiths, 1999
Email: mail@paulgriffiths.net

```
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include "finance.h"

/* Global macros */

#define MAX_BUFFER          (100)
#define ERR_BADVAL         (1)
#define ERR_NOVAL          (2)
#define ERR_BADTYPE        (3)

/* Forward function declarations */

void ParseCmdLine (int argc, char *argv[]);
double GetOptVal (char *arg, int n);
int GetOptType (char *arg);
void OptErr (int n, int errcode);
void ShowHelpMessage();
void ShowVars ();

/* Global variables */

double Stock, Strike, SD, Rate, Time;
int Type;
char variables[][25] = { "Stock price",
                        "Strike price",
                        "Standard deviation",
                        "Risk-free rate of
return",
                        "Time to expiration",
                        "Option type" };

/* main() function */

int main(int argc, char *argv[]) {
    ParseCmdLine(argc, argv);
    ShowVars ();

    printf("%-25s: %14.4f\n\n", "Price",
OptionPrice(Stock, Strike,
```



```

        if ( !optFlags[count] )
            OptErr(count, ERR_NOVAL);
    }

/* General purpose error message function */
void OptErr(int n, int errcode) {

    /* Switch according to what type of error we
    have */

    switch ( errcode ) {
    case ERR_BADVAL:
        fprintf(stderr, "Error: You did not specify
a valid value for %s.\n",
            variables[n]);
        break;

    case ERR_NOVAL:
        fprintf(stderr, "Error: You did not specify
a value for %s.\n",
            variables[n]);
        break;

    case ERR_BADTYPE:
        fprintf(stderr, "Error: You did not specify
a valid type.\n");
        break;
    }
    exit(EXIT_FAILURE);
}

/* Function gets a double value from a command
line switch */

double GetOptVal(char *arg, int n) {
    char    temp[MAX_BUFFER] = {0}, *endptr;
    int     count = 2;
    double  result;

    /* Store all but the first two characters of
    switch
    in temp[], to leave us with just the
    numerical part,
    then convert it to a double.
    */

    while ( arg[count] && count < (MAX_BUFFER +
1) )
        temp[count-2] = arg[count++];
    result = strtod(temp, &endptr);
}

```

```

        /* Check that the user entered a valid
number */

        if ( endptr[0] )
            OptErr(n, ERR_BADVAL);
        return result;
    }

/* Function determines option type from -o
command line switch */

int GetOptType(char *arg) {
    char temp[MAX_BUFFER] = {0};
    int count = 2;

    /* Copy all but the first two characters of
switch into temp[], converting to upper
case as we go. */

    while ( arg[count] && (count < (MAX_BUFFER +
1)) ) {
        temp[count-2] = toupper(arg[count]);
        ++count;
    }

    /* Check that the user specified a valid
type */

    if ( !strncmp(temp, "CALL", 4) )
        return EUROPEAN_CALL;
    else if ( !strncmp(temp, "PUT", 3) )
        return EUROPEAN_PUT;
    else
        OptErr(0, ERR_BADTYPE);
}

/* Shows the help/usage message */

void ShowHelpMessage() {
    printf("\nOption Price Calculator\n");
    printf("=====\n");
    printf("(c) Paul Griffiths 1999\n\n");

    printf("Usage:\n");
    printf("    option -s{n} -x{n} -d{n} -r{n} -
t{n} -o{call,put} [-h]\n\n");
    printf("Switches:\n");
    printf("    -s | -S : Price of underlying
stock\n");
    printf("    -x | -X : Strike/Exercise price
of option\n");
    printf("    -d | -D : Standard deviation of
price of\n");
}

```

```

        printf("                underlying stock per
time period,\n");
        printf("                expressed as a
decimal percentage\n");
        printf("    -r | -R : Risk-free rate of
return per time period,\n");
        printf("                expressed as a
decimal percentage\n");
        printf("    -t | -T : Number of time
periods to expiration\n");
        printf("    -o | -O : Type of option, can
be \"call\" or \"put\"\n");
        printf("    -h | -H : Show this help
message\n\n");

        printf("Example:\n");
        printf("    For a call option on a stock with
a price of $42, which\n");
        printf("    has a standard deviation of 20%%
per year, when the\n");
        printf("    risk-free rate of return is 10%%
per year, the strike\n");
        printf("    price is $40, and the option has
six months to expiration,\n");
        printf("    use:\n\n");
        printf("    option -s42 -x40 -s0.2 -r0.1
-t0.5 -ocall\n\n");

        exit(EXIT_SUCCESS);
}

/* Confirms the variables the user supplied
before showing the price */

void ShowVars() {
    printf("\nOption Price Calculator\n");
    printf("=====\n\n");

    printf("%-25s: %14.4f\n", variables[0],
Stock);
    printf("%-25s: %14.4f\n", variables[1],
Strike);
    printf("%-25s: %14.4f\n", variables[2], SD);
    printf("%-25s: %14.4f\n", variables[3],
Rate);
    printf("%-25s: %14.4f\n", variables[4],
Time);
    printf("%-25s: %14s\n\n", variables[5],
(Type == EUROPEAN_CALL) ? "European call"
: "European put");
}

```

finance.h

```
/*
  Finance.h
  =====

  Interface to option pricing functions
*/

#ifndef PG_FINANCE_H
#define PG_FINANCE_H

#define EUROPEAN_CALL          1
#define EUROPEAN_PUT          2
#define AMERICAN_CALL         3
#define AMERICAN_PUT          4

#define OPTERROR_TYPE          1

double OptionPrice(double Stock, double Strike,
double SD,
                    double Rate, double Time, int
Type);
double CND(double n);
double D(double Stock, double Strike, double SD,
          double Rate, double Time, int Flag);

#endif /* PG_FINANCE_H */
```

finance.c

```
/*
  Finance.c
  =====

  Paul Griffiths, 1999

  Library of financial functions
```

```

*/

#include "finance.h"
#include <math.h>

/*

OptionPrice()
=====

Calculates the price of an option using the
Black-Scholes formula.

Arguments:
    double Stock      : Current price of
underlying stock
    double Strike     : Strike or exercise price
of option
    double SD         : Standard deviation of
stock price per time
                                period, expressed as a
decimal percentage
    double Rate       : Risk-free rate of return
per time period,
                                expressed as a decimal
percentage
    double Time       : Number of time periods
till expiration
    int Type          : Type of option, can be:
                                EUROPEAN_CALL
                                EUROPEAN_PUT
                                AMERICAN_CALL
[UNSUPPORTED]
                                AMERICAN_PUT
[UNSUPPORTED]

Returns:
    The calculated price of the option

*/

double OptionPrice(double Stock, double Strike,
double SD,
                                double Rate, double Time, int
Type) {

    switch ( Type ) {
    case EUROPEAN_CALL:
    case AMERICAN_CALL:
        return Stock * CND(D(Stock, Strike, SD,
Rate, Time, 1)) -
                Strike * exp(-Rate * Time) *
                CND(D(Stock, Strike, SD, Rate, Time,
2));

    case EUROPEAN_PUT:

```

```

        case AMERICAN_PUT:
            return Strike * exp(-Rate * Time) *
                CND(-D(Stock, Strike, SD, Rate, Time,
2)) -
                Stock * CND(-D(Stock, Strike, SD, Rate,
Time, 1));

        default:
            return OPTERROR_TYPE;
    }
}

```

/*

CND()
=====

Calculates the cumulative normal probability
distribution
of (x), accurate to six decimal places.

Arguments:
double x : Observation of a variable which
has a mean of 0
and a standard deviation of 1

Returns:
The cumulative normal probability
distribution of x

*/

```

double CND(double x) {
    int i;
    double a[5] = { 0.319381530, -0.356563782,
1.781477937,
                    -1.821255978, 1.330274429 };
    double result = 0, k = 1 / (1 + 0.2316419 *
fabs(x));

    for ( i = 0; i < 5; i++ ) result += a[i] *
pow(k, i+1);
    result = 1 - result * 1/sqrt(2*3.141592654) *
exp(-(x*x)/2);

    if ( x >= 0 ) return result;
    else return (1 - result);
}

```

/*

D()
===

Calculates d1 and d2 in the Black-Scholes formula

Arguments:

- double Stock : Price of underlying stock
- double Strike : Strike or exercise price of option
- double SD : Standard deviation of stock price per time period, expressed as a decimal percentage
- double Rate : Risk-free rate of return per time period, expressed as a decimal percentage
- double Time : Number of time periods to expiration
- int Flag : Can be 1 or 2, depending on whether d1 or d2 is desired

Returns:
d1 or d2, depending on the value of (Flag)

*/

```
double D(double Stock, double Strike, double SD,
double Rate, double Time, int Flag) {
double result;

if ( Flag == 1 )
result = Rate + (SD*SD)/2;
else if ( Flag == 2 )
result = Rate - (SD*SD)/2;
else
return 0;

return (log(Stock/Strike) + result*Time) /
(SD*sqrt(Time));
}
```

Towers of Hanoi

Function

This program uses a recursive function to output the moves necessary to solve the "Towers of Hanoi" ancient Chinese puzzle, and is a favourite exercise for C instructors.

For those unfamiliar with the puzzle, it involves having three columns on which are placed a number of different sized disks. Originally, there was a stack of 64 disks on one column in order of size (the biggest disk on the bottom of the column, the smallest disk on the top). The ancient Chinese monks reputedly had to move the entire stack from that column to another column, with the following restrictions:

- Only one disk can be moved at a time
- A disk can only be moved onto an empty column, or on top of a larger disk

Programming Issues

This is a problem for which recursion is ideal. Beginners are often surprised at how simple the algorithm is, and how little it appears to do.

To demonstrate how the algorithm works, let us suppose we wish to move 5 disks from column 1 to column 3. The steps are as follows:

1. Move 4 disks from column 1 to column 2
2. Move 1 disk from column 1 to column 3
3. Move 4 disks from column 2 to column 3

In other words, to move n disks from one column to another, move $n-1$ disks to the "spare" column, move the n th disk to the destination column, and finally move the $n-1$ disks from the "spare" column back on top of the n th disk. Recursion then takes care of moving the $n-1$ disks by using the same algorithm until we have no more disks left to move.

The program has been complicated slightly more than necessary by adding a command line interface to set the parameters.

Usage

Issue the following command line:

```
hanoi -d4 -s1 -e3
```

where:

- `-d{n}` specifies the number of disks to move
- `-s{n}` specifies the column to move from (must be 1, 2 or 3)
- `-e{n}` specifies the column to move to (must be 1, 2 or 3 and different to the start column)

hanoi.c

```
/*  
  
    HANOI.C  
    =====  
    (c) Paul Griffiths 1999  
    Email: mail@paulgriffiths.net  
  
    Outputs moves necessary to complete the  
    "Towers of Hanoi" puzzle using a recursive  
    function.  
  
*/  
  
#include <stdlib.h>  
#include <string.h>  
#include <stdio.h>  
  
/* Forward function declarations */  
  
void Hanoi          (int StartCol, int EndCol, int  
nDisks);  
int ParseCmdLine(int argc, char *argv[],  
                int *StartCol, int *EndCol, int  
*nDisks);  
  
/* main() function */  
  
int main(int argc, char *argv[]) {  
    int StartCol, EndCol, nDisks;  
  
    /* Set program variables */  
  
    if ( !ParseCmdLine(argc, argv, &StartCol,  
&EndCol, &nDisks) )  
        exit(EXIT_FAILURE);
```

```

        /* Output moves necessary */

        printf("Towers of Hanoi!\n");
        printf("=====\n\n");

        printf("To move %d disks from column %d to
column %d...\n\n",
            nDisks, StartCol, EndCol);
        Hanoi(StartCol, EndCol, nDisks);

        exit(EXIT_SUCCESS);
    }

/* Output to stdin the moves necessary to move a
stack
of nDisks disks from column StartCol to
column EndCol,
according to the rules of the puzzle.
*/

void Hanoi(int StartCol, int EndCol, int nDisks)
{
    if ( nDisks ) {
        Hanoi(StartCol, 6 - (StartCol + EndCol),
nDisks - 1);
        printf("Move disk from column %d to column
%d\n", StartCol, EndCol);
        Hanoi(6 - (StartCol + EndCol), EndCol,
nDisks - 1);
    }
}

/* Set program variables according to command
line options */

int ParseCmdLine(int argc, char *argv[],
                int *StartCol, int *EndCol, int
*nDisks) {
    int n = 1;

    /* Set default values for variables */

    *nDisks    = 4;
    *StartCol  = 1;
    *EndCol    = 2;

    /* Set variables according to command line
options */

    while ( n < argc ) {

```

```

        if ( !strncmp(argv[n], "-d", 2) ||
!strncmp(argv[n], "-D", 2) ) {
            *nDisks = strtol(&argv[n][2], NULL, 0);
            if ( *nDisks < 1 ) {
                printf("Must have at least 1 disk to
move!\n");
                return 0;
            }
        }
        else if ( !strncmp(argv[n], "-s", 2) ||
!strncmp(argv[n], "-S", 2) ) {
            *StartCol = strtol(&argv[n][2], NULL,
0);
            if ( *StartCol < 1 || *StartCol > 3 ) {
                printf("Start column must be 1, 2 or
3!\n");
                return 0;
            }
        }
        else if ( !strncmp(argv[n], "-e", 2) ||
!strncmp(argv[n], "-E", 2) ) {
            *EndCol = strtol(&argv[n][2], NULL, 0);
            if ( *EndCol < 1 || *EndCol > 3 ) {
                printf("End column must be 1, 2 or
3!\n");
                return 0;
            }
            if ( *EndCol == *StartCol ) {
                printf("Start column and end column
must be different!\n");
                return 0;
            }
        }
        ++n;
    }
    return 1;
}

```

Questions: -

1. Write down program to calculate option price
2. Write down program of tower of hanoi