Semester 2, 2013

Assignment 1

In this project you will demonstrate your understanding of arrays and structures, and functions that make use of them. You will also extend your skills in terms of program design, testing, and debugging.

The Story...

Most modern cryptography relies on doing arithmetic with very large numbers. One example is the RSA algorithm, used by secure websites to prove that the server really is who it’s claiming to be. RSA can also be used for encrypting messages. Either way, its security relies on the assumption that it’s hard to factorize a number that is the product of two large primes, where “large” means hundreds of decimal digits.

The problem is, the data type int in C can only store values up to 231 − 1 = 2,147,483,647. If larger values must be manipulated accurately, a different representation for integers is required. Your task in this project is to develop an *integer calculator* that works with extended-precision integer values. The calculator has a simple interface, and 26 “variables” (or memories) into which values can be stored. For example a session with your calculator (where > is the prompt from the system) might look like:

mac: ./ass1  
> a = 2147483647  
>a?  
2147483647  
>a+1  
>a?  
2147483648  
> b = 999999999999999  
> c = 1000000000000000000 >c+b  
>c?  
1000999999999999999  
>b?  
999999999999999  
>a?  
2147483648  
> ^D  
mac:

Note the extremely limited *syntax*: constants or other variables can be assigned to variables using an “=” operator; or variable values can be altered using a single operator and another variable or constant (and “+” is the only one implemented in the skeleton; or variable values can be printed using “?”. The variable in question is always the first thing specified on each input line; then a single operator; and then (for most operators) either another variable name, or an extended-precision integer constant.

To allow storage of very large integers, your calculator needs to make use arrays of ASCII characters, with one decimal digit stored per character in the array. You are to design a suitable structure and

1

representation, assuming that values of up to INTSIZE digits are to be handled. Other information might also be required in each “number”, including a count of the number of digits, and a sign.

The expected input format for all stages is identical – a sequence of simple calculator commands of the type shown above.

Stage 1 – Getting Started (marks up to 8/15)

Obtain a copy of the skeleton file using the link on the LMS, and spend some time (hours!) studying the way it is constructed, including compiling and executing it. It only works with int variables at present, but is already quite a complex program!

Note the use of the type longint t. Each of the longint t variables in the array vars stores one integer, with 26 variables available in total, named ’a’ to ’z’ (and indexed from 0 to 25, via the function to varnum). These variables are changed by various functions, depending on the commands read from the input. There is no subtraction, but negative numbers are available via negative constants:

> a = 10 > a + -20 >a?  
-10

In this first stage you are to design and implement an alternative data structure for longint t. You should use an array of INTSIZE characters for each longint t, together with the other required infor- mation, all bound together as a struct (Chapter 8).

You will need to make non-trivial changes in many of the functions (and perhaps introduce new ones), and need to be highly systematic in the way you approach this task. You should plan carefully, rather than just leaping in and starting to edit the file. Then, before moving through the rest of the Stages, you should test your program carefully, to make sure that you have retained all of the existing functionality. And remember that if you truly scramble things up, you can always recopy the skeleton and start again.

Stage 2 – User Friendly Input and Output (marks up to 10/15)

Alter the program so that numbers are always printed with commas (Australian-style, every three digits), and can also (optionally) have (correctly or incorrectly placed) commas in them when they are entered:

> a = 2147483647 >a?  
2,147,483,647  
> b = 2,147,483,647 >b?

2,147,483,647  
> c = -21,47483647 >c? -2,147,483,647

Stage 3 – Another Operator (marks up to 15/15)

Now add in a multiplication operator, \*:

> a = 12,345,654,321  
> a \* 1,234,321  
>a? 15,238,500,387,151,041

page2image838301760page2image838302016page2image838302272page2image838302528page2image838302848

2

You will need to think carefully about the *algorithm* that you will use in this task. Best bet is to implement some functions that will allow you to carry out the calculation the way that you learned to do long multiplication at school. For example, a function times ten that multiplies a number by ten might be useful; as might a function times digit, which takes a longint t and multiplies it by a single-digit number.

There are also other ways (some of them much more efficient than the “school” algorithm) to doing multiplication, and you can be inventive if you wish. Just don’t try and do it by repeated addition.

Stage 4 – Yet More Operators (marks up to 15/15)

Of course, you know what is coming next – an integer “power of” operator, “^”:

>a=2  
> a ^ 100  
>a?  
> 1,267,650,600,228,229,401,496,703,205,376

If you are successful in this stage, you will be able to earn back one or two of the marks you lost in the earlier stages (assuming that you lost some, you can’t go past 15/15 in total). But *please please please* don’t start on this task until your program through to Stage 3 is (in your opinion) perfect, and is safely submitted, and is verified, and is backed up somewhere. (Still keen? Add in division too, using “/”.)

The boring stuff...

This project is worth 15% of your final mark. You don’t have to do Stage 4 in order to get 15/15.  
You need to submit your program for assessment; detailed instructions on how to do that will be posted on the LMS once submissions are opened. You will need to log in to a Unix server and submit your files to a software system known as submit. You can (and should) use submit both early and often – to get used to the way it works, and also to check that your program compiles correctly on our test system, which has some different characteristics to the lab machines. Only the last submission that

you make before the deadline will be marked.  
You may discuss your design/plans during your workshop, and with others in the class, but what gets

typed into your program must be individual work, not from anyone else. So, do not give hard copy or soft copy of your work to anyone else; do not “lend” your memory stick to others; and do not ask others to give you their programs “just so that I can take a look and get some ideas, I won’t copy, honest”. The best way to help your friends in this regard is to say “are you out of your mind?” if they ask for a copy of, or to see, your program, pointing out that your refusal, and their acceptance of it, is the only thing that will preserve your friendship. *A sophisticated program that undertakes deep structural analysis of C code identifying regions of similarity will be run over all submissions in “compare every pair” mode.*

Deadline: Programs not submitted by 6:00pm on Friday 20 September will lose penalty marks at the rate of two marks per day or part day late. Students seeking extensions for medical or other “outside my control” reasons should email Alistair, ammoffat@unimelb.edu.au.

*And remember, algorithms are fun!*

Solution：

/\* Extended precision integer calculator program

\* Implements + and \* operations

\*

\* Sample solution written by Alistair Moffat, October 2013

\*

\*/

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <strings.h>

#include <ctype.h>

#define INTSIZE 100 /\* max number of digits per integer value \*/

#define LINELEN 102 /\* maximum length of any input line \*/

#define NVARS 26 /\* number of different variables \*/

#define CH\_BLANK ' ' /\* character blank \*/

#define CH\_ZERO '0' /\* character zero \*/

#define CH\_ONE '1' /\* character one \*/

#define CH\_NINE '9' /\* character nine \*/

#define CH\_A 'a' /\* character 'a', first variable name \*/

#define CH\_PL '+' /\* character '+' \*/

#define CH\_NE '-' /\* character '-' \*/

#define CH\_COM ',' /\* character ',' \*/

#define INT\_TEN 10 /\* integer 10 \*/

#define ERROR (-1) /\* error return value from some functions \*/

#define PRINT '?' /\* the print operator \*/

#define ASSIGN '=' /\* the assignment operator \*/

#define PLUS '+' /\* the addition operator \*/

#define MULT '\*' /\* the multiplication operator \*/

#define POWER '^' /\* the power-of operator \*/

#define ALLOPS "?=+\*^" /\* list of all valid operators \*/

#define SGNCHRS "+-" /\* the two sign characters \*/

#define NUMCHRS ",0123456789"

/\* list of characters legal within numbers \*/

#define POSI 0 /\* indication of positive number \*/

typedef struct {

int nega; /\* zero for positive, non-zero for negative \*/

int ndig;

char digs[INTSIZE];

} longint\_t;

/\* one extended-precision "variable" \*/

#define LONG\_ZERO {0,POSI,{0}}

/\* and an initializer for variables that are zero \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* function prototypes \*/

char add\_to\_digit(char d1, int val, int \*carry);

void do\_assign(longint\_t \*var1, longint\_t \*var2);

void do\_mult(longint\_t \*var1, longint\_t \*var2);

void do\_plus(longint\_t \*var1, longint\_t \*var2);

void do\_power(longint\_t \*var1, longint\_t \*var2);

void do\_print(longint\_t \*var);

int get\_second\_value(longint\_t vars[], char \*rhsarg, longint\_t \*second\_value);

int mag\_cmp(longint\_t \*var1, longint\_t \*var2);

void mult\_one\_digit(longint\_t \*var, int digit);

void mult\_ten(longint\_t \*var);

void overflow\_abort(void);

longint\_t parse\_num(char \*rhs);

void print\_prompt(void);

void process\_line(longint\_t vars[], char \*line);

int read\_line(char \*line, int maxlen);

void simple\_plus(longint\_t \*var1, longint\_t \*var2);

void simple\_suba(longint\_t \*var1, longint\_t \*var2);

char sub\_digit(char d1, int val, int \*borrow);

char to\_digit(int number);

int to\_int(char digit);

int to\_varnum(char ident);

void zero\_vars(longint\_t vars[]);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* main program controls all the action

\*/

int

main(int argc, char \*\*argv) {

char line[LINELEN+1];

longint\_t vars[NVARS];

zero\_vars(vars);

print\_prompt();

while (read\_line(line, LINELEN)) {

if (strlen(line)>0) {

/\* non empty line, so process it \*/

process\_line(vars, line);

}

print\_prompt();

}

/\* all done, so pack up and go home \*/

printf("\n");

return 0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* prints the prompt indicating ready for input

\*/

void

print\_prompt(void) {

printf("> ");

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* read a line of input into the array passed as argument

\* returns false if there is no input available

\* all whitespace characters are removed

\*/

int

read\_line(char \*line, int maxlen) {

int i=0, c;

while (((c=getchar())!=EOF) && (c!='\n')) {

if (i<maxlen && !isspace(c)) {

line[i++] = c;

}

}

line[i] = '\0';

return ((i>0) || (c!=EOF));

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* process a command by parsing the input line into parts

\*/

void

process\_line(longint\_t vars[], char \*line) {

int varnum, optype, status;

longint\_t second\_value;

/\* determine the LHS variable, it

\* must be first character in line

\*/

varnum = to\_varnum(line[0]);

if (varnum==ERROR) {

printf("Invalid LHS variable\n");

return;

}

/\* more testing for validity

\*/

if (strlen(line)<2) {

printf("No operator supplied\n");

return;

}

/\* determine the operation to be performed, it

\* must be second character in line

\*/

optype = line[1];

if (strchr(ALLOPS, optype) == NULL) {

printf("Unknown operator\n");

return;

}

/\* determine the RHS argument (if one is required),

\* it must start in third character of line

\*/

if (optype != PRINT) {

if (strlen(line)<3) {

printf("No RHS supplied\n");

return;

}

status = get\_second\_value(vars, line+2, &second\_value);

if (status==ERROR) {

printf("RHS argument is invalid\n");

return;

}

}

/\* finally, do the actual operation

\*/

if (optype == PRINT) {

do\_print(vars+varnum);

} else if (optype == ASSIGN) {

do\_assign(vars+varnum, &second\_value);

} else if (optype == PLUS) {

do\_plus(vars+varnum, &second\_value);

} else if (optype == MULT) {

do\_mult(vars+varnum, &second\_value);

} else if (optype == POWER) {

do\_power(vars+varnum, &second\_value);

}

return;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* convert a character variable identifier to a variable number

\*/

int

to\_varnum(char ident) {

int varnum;

varnum = ident - CH\_A;

if (0<=varnum && varnum<NVARS) {

return varnum;

} else {

return ERROR;

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* process the input line to extract the RHS argument, which

\* should start at the pointer that is passed

\*/

int

get\_second\_value(longint\_t vars[], char \*rhsarg,

longint\_t \*second\_value) {

char \*p;

int varnum2;

if (strchr(NUMCHRS, \*rhsarg) != NULL ||

strchr(SGNCHRS, \*rhsarg) != NULL) {

/\* first character is a digit or a sign, so RHS

\* should be a number

\*/

p = rhsarg+1;

while (\*p) {

if (strchr(NUMCHRS, \*p) == NULL) {

/\* nope, found an illegal character \*/

return ERROR;

}

p++;

}

\*second\_value = parse\_num(rhsarg);

return !ERROR;

} else {

/\* argument is not a number, so might be a variable \*/

varnum2 = to\_varnum(\*rhsarg);

if (varnum2==ERROR || strlen(rhsarg)!=1) {

/\* nope, not a variable either \*/

return ERROR;

}

/\* is a variable, so can use its value to assign to

\* second\_value

\*/

\*second\_value = vars[varnum2];

return !ERROR;

}

return ERROR;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* create an internal-format number out of a string

\*/

longint\_t

parse\_num(char \*rhs) {

int i=0;

longint\_t num = LONG\_ZERO;

char t;

while (\*rhs && \*rhs==CH\_BLANK) {

rhs++;

}

if (\*rhs==CH\_NE) {

num.nega = !POSI;

rhs++;

}

if (\*rhs==CH\_PL) {

rhs++;

}

while (\*rhs) {

if (\*rhs==CH\_COM || \*rhs==CH\_BLANK) {

rhs++;

} else {

if (i!=0 || \*rhs!=CH\_ZERO) {

if (i==INTSIZE) {

overflow\_abort();

}

num.digs[i++] = \*rhs;

}

rhs++;

}

}

num.ndig = i;

/\* now reverse so that low order digit is in first position \*/

for (i=0; i<num.ndig/2; i++) {

t = num.digs[i];

num.digs[i] = num.digs[num.ndig-1-i];

num.digs[num.ndig-1-i] = t;

}

return num;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* convert a character digit to the int equivalent, but null bytes

\* stay as zero integers

\*/

int

to\_int(char digit) {

if (digit != '\0') {

return digit - CH\_ZERO;

} else {

return 0;

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* and back again to a digit

\*/

char

to\_digit(int number) {

return number + CH\_ZERO;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* print out a longint value

\*/

void

do\_print(longint\_t \*var) {

int i;

if (var->ndig==0) {

/\* special case for zero \*/

printf("0\n");

return;

}

if (var->nega) {

printf("-");

}

for (i=var->ndig-1; i>=0; i--) {

printf("%c", var->digs[i]);

if (i%3==0 && i>0) {

printf("%c", CH\_COM);

}

}

printf("\n");

return;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* update the indicated variable var1 by doing an assignment

\* using var2

\*/

void

do\_assign(longint\_t \*var1, longint\_t \*var2) {

/\* hey, this is trivial in this representation

\*/

\*var1 = \*var2;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* add val to the character and the possible carry to get a new

\* character as their sum, and a new integer carry

\*/

char

add\_to\_digit(char d1, int val, int \*carry) {

int d;

d = d1 + val + \*carry;

\*carry = 0;

while (d > CH\_NINE) {

d -= INT\_TEN;

\*carry += 1;

}

return (char)d;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* substract val to the character and also the borrow to get a new

\* character as their sum, and a new integer borow

\*/

char

sub\_digit(char d1, int val, int \*borrow) {

int d;

d = d1 - val - \*borrow;

\*borrow = 0;

if (d<CH\_ZERO) {

d += INT\_TEN;

\*borrow = 1;

}

return (char)d;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* update the indicated variable var1 by doing an addition

\* using var2 to compute var1 = var1 + var2

\*/

void

do\_plus(longint\_t \*var1, longint\_t \*var2) {

longint\_t temp;

if (var1->nega == var2->nega) {

/\* same signs, can just add \*/

simple\_plus(var1, var2);

} else {

/\* different signs, need to get them right way round \*/

temp = \*var2;

if (mag\_cmp(var1, var2)>=0) {

/\* can process in this order \*/

simple\_suba(var1, &temp);

} else {

/\* need to reverse the order \*/

simple\_suba(&temp, var1);

\*var1 = temp;

}

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* return negative if magnitude of first arg is smaller than second,

\* zero if first arg has same magnitude as second

\* positive if magnitude of first arg is greater than second

\*/

int

mag\_cmp(longint\_t \*var1, longint\_t \*var2) {

int i;

if (var1->ndig != var2->ndig) {

return var1->ndig - var2->ndig;

}

for (i=var1->ndig-1; i>=0; i--) {

if (var1->digs[i] != var2->digs[i]) {

return var1->digs[i] - var2->digs[i];

}

}

return 0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* simple plus situation where var1 and var2 have the same sign

\*/

void

simple\_plus(longint\_t \*var1, longint\_t \*var2) {

int i, carry=0, ndig;

/\* insert zeros to make the two numbers the same length \*/

ndig = var1->ndig;

if (var2->ndig > ndig) {

ndig = var2->ndig;

}

for (i=var1->ndig; i<ndig; i++) {

var1->digs[i] = CH\_ZERO;

}

for (i=var2->ndig; i<ndig; i++) {

var2->digs[i] = CH\_ZERO;

}

/\* do the addition over ndig digits \*/

for (i=0; i<ndig; i++) {

var1->digs[i] = add\_to\_digit(var1->digs[i],

to\_int(var2->digs[i]), &carry);

}

/\* been a carry out of the top digit? \*/

if (carry) {

if (i>=INTSIZE) {

/\* nope, would overflow arrays \*/

overflow\_abort();

}

var1->digs[i++] = CH\_ONE;

}

/\* number of digits in the output \*/

var1->ndig = i;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* simple subtraction situation where var2 is less than var1

\* and can be subtracted from it

\*/

void

simple\_suba(longint\_t \*var1, longint\_t \*var2) {

int i, borrow=0, ndig;

/\* put zeros in to second argument to match first \*/

for (i=var2->ndig; i<var1->ndig; i++) {

var2->digs[i] = CH\_ZERO;

}

ndig = i;

/\* do the subtraction \*/

for (i=0; i<ndig; i++) {

var1->digs[i] = sub\_digit(var1->digs[i],

to\_int(var2->digs[i]), &borrow);

}

/\* check for emergence of leading zeros \*/

while (ndig>0 && var1->digs[ndig-1]==CH\_ZERO) {

ndig--;

}

var1->ndig = ndig;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* multiply argument by ten by shifting, and adding a zero

\*/

void

mult\_one\_digit(longint\_t \*var, int digit) {

longint\_t working=LONG\_ZERO;

int i, carry;

carry = 0;

for (i=0; i<var->ndig; i++) {

working.digs[i] = add\_to\_digit(CH\_ZERO,

to\_int(var->digs[i])\*digit,

&carry);

}

if (carry) {

if (i==INTSIZE) {

overflow\_abort();

}

working.digs[i++] = to\_digit(carry);

}

working.ndig = i;

\*var = working;

return;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* multiply argument by ten by shifting, and adding a zero

\*/

void

mult\_ten(longint\_t \*var) {

int i;

if (var->ndig==INTSIZE) {

overflow\_abort();

}

for (i=var->ndig; i>0; i--) {

var->digs[i] = var->digs[i-1];

}

var->digs[0] = CH\_ZERO;

var->ndig++;

return;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* update the indicated variable var1 by doing a multiplication

\* using var2 to compute var1 = var1 \* var2

\*/

void

do\_mult(longint\_t \*var1, longint\_t \*var2) {

longint\_t product = LONG\_ZERO;

longint\_t oneterm;

int i;

for (i=var2->ndig-1; i>=0; i--) {

/\* process one digit from var2 \*/

oneterm = \*var1;

mult\_one\_digit(&oneterm, to\_int(var2->digs[i]));

mult\_ten(&product);

do\_plus(&product, &oneterm);

}

/\* now sort out the sign \*/

product.nega = var1->nega != var2->nega;

\*var1 = product;

return;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* update the indicated variable var1 by raising it to the

\* power of variable var2, var1 = var1 ^ var2

\*/

void

do\_power(longint\_t \*var1, longint\_t \*var2) {

printf("Power-of operator not implemented\n");

return;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* set the vars array to all zero values

\*/

void

zero\_vars(longint\_t vars[]) {

int i;

longint\_t zero = LONG\_ZERO;

for (i=0; i<NVARS; i++) {

do\_assign(vars+i, &zero);

}

return;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* exit if numbers are going to get too big

\*/

void

overflow\_abort() {

fprintf(stderr, "Integer overflow, program terminated\n");

exit(EXIT\_FAILURE);

}