♀
```
 ---------------------------------------------------
         Edward Bolles
         CS 620
         Sun Oct 21 2007  07:49:42
 ---------------------------------------------------

EXTENDED PRECISION INTEGER ARITHMETIC

Type in the operands in the math problem.
The answer will be returned.

+ = add
- = subtract
* = multiply
? = compare

----------------------------------------

Enter your expression:
999 + 1
answer is +1,000
Try again? y/n
y
----------------------------------------

Enter your expression:
1 - 1,000
answer is -999
Try again? y/n
y
----------------------------------------

Enter your expression:
-999 + -1
answer is -1,000
Try again? y/n
y
----------------------------------------

Enter your expression:
999 * 2
answer is +1,998
Try again? y/n
y
----------------------------------------
```

```
Enter your expression:
2 * 999
answer is +1,998
Try again? y/n
y
----------------------------------------

Enter your expression:
-1,OOO * 1OOO
answer is -1,OOO,OOO
Try again? y/n
y
----------------------------------------

Enter your expression:
-999 ? 1
-999 < 1
Try again? y/n
```

```
/*****************************************************/
/* linked.c */
/* linked arithmetic */
/* Edward Bolles            */
/*****************************************************/

#include "tools.h"

typedef struct node{
        int digit;
        struct node *next;
        struct node *prev;
} node;

typedef enum{
        false,
        true
} bool;


/*-----------prototypes---------------*/
void enter_data(char* first, char* oper, char* sec);
node* newbigint(char* string);
void select_mult(node *num1, node *num2, node *num3);
void select_oper(node *num1, node *num2, node *num3, char oper);
void select_comp(node *num1, node *num2, char *first, char *sec);
int compare(node *num1, node *num2);
void oper_add(node *num1, node *num2, node *num3);
void oper_sub(node *num1, node *num2, node *num3);
bool recurseb(node *num1);
void oper_mult(node *num1, node *num2, node *num3);
void printbigint(node *num3p);
/*-----------main---------------*/
void main (void)
{
        char* first;
        char oper;
        char* sec;
        char again;

        node* num1;
        node* num2;
        node* num3;

        first = malloc( 20 * sizeof( char ));
        if (first == NULL) fatal( "Out of memory." );
```

```
        sec = malloc( 20 * sizeof( char ));
        if (sec == NULL) fatal( "Out of memory." );



        banner();
        puts( "\nEXTENDED PRECISION INTEGER ARITHMETIC\n" );
        printf( "Type in the operands in the math problem.\n"
           "The answer will be returned.\n\n");
        printf("+ = add\n- = subtract\n* = multiply\n? = compare\n\n");
        for(;;)
        {
                printf("----------------------------------------\n\n");

                printf("Enter your expression:\n");
                enter_data(first, &oper, sec);
                num1 = newbigint(first);
                num2 = newbigint(sec);
                num3 = malloc(sizeof(node));
                num3->next = num3;
                num3->prev = num3;

                if (oper == '+' || oper == '-') select_oper(num1, num2, num3, oper);
                if (oper == '*') select_mult(num1, num2, num3);
                if (oper == '?') select_comp(num1, num2, first, sec);

                printf("Try again? y/n\n");
                scanf(" %c", &again);
                if ('N' == toupper(again)) break;
        }
        bye();
}



//--------------------------------------
//enter data from keyboard into letters array
void enter_data(char* first, char* operp, char* sec)
{
        char buffer[40];
        int len;
        char oper;

        //scan first operand
        scanf("%s", buffer );
        len = strlen( buffer );//measure length
```

```
        first = realloc( first, len * sizeof( char ));
        if (first == NULL) fatal( "Out of memory." );
        strcpy( first, buffer );//copy string
        strcat( first, "\0" );//add null terminator

        //scan operator
        scanf(" %c", &oper);

        //scan second operand
        scanf("%s", buffer );
        len = strlen( buffer );//measure length
        sec = realloc( sec, len * sizeof( char ));
        if (sec == NULL) fatal( "Out of memory." );
        strcpy( sec, buffer );//copy string
        strcat( sec, "\0" );//add null terminator

        *operp = oper;


}


//---------------------------------------
//
node* newbigint(char* string)
{
        struct node* head;
        struct node* newnode;
        int len;
        int slot = 0;

        head = malloc(sizeof(node)); // allocate
        head->next = head;
        head->prev = head;
        if (string[slot] == '-')
        {
                head->digit = 101;
                slot++; //101 means negative, advance to read digits
        }
        else if (string[slot] == '+')
        {
                head->digit = 100;
                slot++; //100 means positive, advance to read digits
        }
        else head->digit = 100;//default to positive if no sign given, read as digit instead

        len = strlen(string);
```

```
        //insert new digit links
        for (; slot < len; ++slot)//scan from most signif digit to least, left to right
        {
                if (string[slot] != ',')
                {
                newnode = malloc(sizeof(node)); // allocate
                newnode->digit = string[slot] - 48;// convert to integer, assign value
                newnode->next = head->next;//point newnode to first in list
                head->next = newnode;//point head to newnode
                newnode->prev = head;
                newnode->next->prev = newnode;
                }
        }
        return head;
}

//-------------------------------------
//
void select_oper(node *num1, node *num2, node *num3, char oper)
{
        int comp;
        comp = compare(num1, num2);

        //select add operation
        if ((num1->digit == 100) && oper == '+' && (num2->digit == 100))
        {
                num3->digit = 100;
                oper_add(num1, num2, num3);
                printbigint(num3);
        }
        else if ((num1->digit == 100) && oper == '-' && (num2->digit == 101))
        {
                num3->digit = 100;
                oper_add(num1, num2, num3);
                printbigint(num3);
        }
        else if ((num1->digit == 101) && oper == '-' && (num2->digit == 100))
        {
                num3->digit = 101;
                oper_add(num1, num2, num3);
                printbigint(num3);
        }
        else if ((num1->digit == 101) && oper == '+' && (num2->digit == 101))
        {
                num3->digit = 101;
                oper_add(num1, num2, num3);
```

```
        printbigint(num3);
}

//select subtract operation
else if ((num1->digit == 1OO) && oper == '+' && (num2->digit == 1O1) && comp >= 0)
{
        num3->digit = 1OO;
        oper_sub(num1, num2, num3);
        printbigint(num3);
}
else if ((num1->digit == 1OO) && oper == '+' && (num2->digit == 1O1) && comp == -1)
{
        num3->digit = 1O1;
        oper_sub(num1, num2, num3);
        printbigint(num3);
}

else if ((num1->digit == 1OO) && oper == '-' && (num2->digit == 1OO) && comp >= 0)
{
        num3->digit = 1OO;
        oper_sub(num1, num2, num3);
        printbigint(num3);
}

else if ((num1->digit == 1OO) && oper == '-' && (num2->digit == 1OO) && comp == -1)
{
        num3->digit = 1O1;
        oper_sub(num1, num2, num3);
        printbigint(num3);
}

else if ((num1->digit == 1O1) && oper == '+' && (num2->digit == 1OO) && comp == 1)
{
        num3->digit = 1O1;
        oper_sub(num1, num2, num3);
        printbigint(num3);
}

else if ((num1->digit == 1O1) && oper == '+' && (num2->digit == 1OO) && comp <= 0)
{
        num3->digit = 1OO;
        oper_sub(num1, num2, num3);
        printbigint(num3);
}

else if ((num1->digit == 1O1) && oper == '-' && (num2->digit == 1O1) && comp == 1)
{
```

```
                num3->digit = 1O1;
                oper_sub(num1, num2, num3);
                printbigint(num3);
        }

        else if ((num1->digit == 1O1) && oper == '-' && (num2->digit == 1O1) && comp <= 0)
        {
                num3->digit = 1OO;
                oper_sub(num1, num2, num3);
                printbigint(num3);
        }
        else printf("selection error\n");
}

//-------------------------------------
//
void select_mult(node *num1, node *num2, node *num3)
{
        //select mult operation
                if ((num1->digit == 1OO) && (num2->digit == 1OO))
                {
                        num3->digit = 1OO;
                        oper_mult(num1, num2, num3);
                        printbigint(num3);
                }
                else if ((num1->digit == 1O1) && (num2->digit == 1O1))
                {
                        num3->digit = 1OO;
                        oper_mult(num1, num2, num3);
                        printbigint(num3);
                }
                else if ((num1->digit == 1OO) && (num2->digit == 1O1))
                {
                        num3->digit = 1O1;
                        oper_mult(num1, num2, num3);
                        printbigint(num3);
                }
                else if ((num1->digit == 1O1) && (num2->digit == 1OO))
                {
                        num3->digit = 1O1;
                        oper_mult(num1, num2, num3);
                        printbigint(num3);
                }
}

//-------------------------------------
//
```

```
void select_comp(node *num1, node *num2, char *first, char *sec)
{
	int comp;
	comp = compare(num1, num2);
	if (comp == 1 && num1->digit == 100 && num2->digit == 100) printf("%s > %s\n", first, sec);
	if (comp == 1 && num1->digit == 101 && num2->digit == 101) printf("%s < %s\n", first, sec);

	if (comp == 0 && num1->digit == 100 && num2->digit == 100) printf("%s = %s\n", first, sec);
	if (comp == 0 && num1->digit == 101 && num2->digit == 101) printf("%s = %s\n", first, sec);

	if (comp == -1 && num1->digit == 100 && num2->digit == 100) printf("%s < %s\n", first, sec);
	if (comp == -1 && num1->digit == 101 && num2->digit == 101) printf("%s > %s\n", first, sec);

	if (num1->digit == 100 && num2->digit == 101)  printf("%s > %s\n", first, sec);
	if (num1->digit == 101 && num2->digit == 100)  printf("%s < %s\n", first, sec);
}

//----------------------------------------
//
int compare(node *num1, node *num2)
{
	int n1 = 0;
	int n2 = 0;
	int len;
	node *travel;

	//translate int array to int
	travel = num1;
	len = 0;
	for (;;)
	{
		travel = travel->next;
		n1 = n1 + ( travel->digit * ( pow(10, len) ));
		if (travel->next->digit > 99) break;
		len++;
	}

	//translate int array to int
	travel = num2;
	len = 0;
	for (;;)
	{
		travel = travel->next;
		n2 = n2 + ( travel->digit * ( pow(10, len) ));
		if (travel->next->digit > 99) break;
		len++;
	}
```

```
        if (n1 > n2) return 1;
        if (n1 == n2) return 0;
        if (n1 < n2) return -1;
}



//--------------------------------------
//
void oper_add(node *num1, node *num2, node *num3)
{

        struct node* newnode;
        struct node* extranode;


        for(;;)
        {
                //prepare next, add places if needed
                if ((num1->next->digit < 99) && (num2->next->digit > 99))
                {

                        extranode = malloc(sizeof(node));
                        extranode->digit = 0;
                        extranode->next = num2->next;
                        num2->next = extranode;
                        extranode->prev = num2;
                        extranode->next->prev = extranode;

                }
                if ((num1->next->digit > 99) && (num2->next->digit < 99))
                {

                        extranode = malloc(sizeof(node));
                        extranode->digit = 0;
                        extranode->next = num1->next;
                        num1->next = extranode;
                        extranode->prev = num1;
                        extranode->next->prev = extranode;

                }

                //advance to next place
                num1 = num1->next;
                num2 = num2->next;

                //operate
```

```
                newnode = malloc(sizeof(node)); // allocate
                newnode->digit = num1->digit + num2->digit;//add digits
                if (newnode->digit > 9) //check for carry
                {
                        if (num1->next->digit > 99) //add place to num1 if detect head
                        {
                                extranode = malloc(sizeof(node));
                                extranode->digit = 0;
                                extranode->next = num1->next;
                                num1->next = extranode;
                                extranode->prev = num1;
                                extranode->next->prev = extranode;
                        }
                        num1->next->digit += newnode->digit / 10; //put any tens place digit into next place
                        newnode->digit %= 10; //put mod as ones place digit in current place
                }
                //insert new digit in answer
                newnode->prev = num3->prev;//point newnode to first in list
                num3->prev = newnode;//point head to newnode
                newnode->next = num3;
                newnode->prev->next = newnode;


                //check for end
                if ((num1->next->digit > 99) && (num2->next->digit > 99))
                {

                        break;
                }
        }

}



//---------------------------------
// subtraction
void oper_sub(node *num1p, node *num2p, node *num3)
{
        bool borrow;
        struct node *num1;
        struct node *num2;
        struct node *newnode;
        struct node *extranode;
        int comp;
```

```
//put greater number on top
comp = compare(num1p, num2p);
if (comp == 1)
{
	num1 = num1p;
	num2 = num2p;

}
if (comp == 0)
{
	num1 = num1p;
	num2 = num2p;

}
if (comp == -1)
{
	num1 = num2p;
	num2 = num1p;

}


for(;;)
{
	//prepare next, add places if needed
	if ((num1->next->digit < 99) && (num2->next->digit > 99))
	{
		extranode = malloc(sizeof(node));
		extranode->digit = 0;
		extranode->next = num2->next;
		num2->next = extranode;
		extranode->prev = num2;
		extranode->next->prev = extranode;
	}
	if ((num1->next->digit > 99) && (num2->next->digit < 99))
	{
		extranode = malloc(sizeof(node));
		extranode->digit = 0;
		extranode->next = num1->next;
		num1->next = extranode;
		extranode->prev = num1;
		extranode->next->prev = extranode;
	}

	//advance to next place
	num1 = num1->next;
```

```
                num2 = num2->next;


                //operate
                newnode = malloc(sizeof(node)); // allocate

                //borrow if need
                if (num1->digit < num2->digit)
                {
                        borrow = recurseb(num1);
                        if (borrow == true)
                        {
                                num1->digit += 10;
                        }
                }

                newnode->digit = num1->digit - num2->digit; // subtraction operation

                //insert new digit in answer
                newnode->prev = num3->prev;//point newnode to first in list
                num3->prev = newnode;//point head to newnode
                newnode->next = num3;
                newnode->prev->next = newnode;

                //check for end
                if ((num1->next->digit > 99) && (num2->next->digit > 99))
                {
                        break;
                }
        }

}

//-----------------------------------
//
bool recurseb(node *num1)
{
        bool borrow;

        num1 = num1->next;

        //base cases could borrow, could not borrow
        if ((num1->digit >= 1) && (num1->digit < 99))
        {
                num1->digit--;
                return true;
        }
```

```
        if (num1->digit > 99 ) return false;

        borrow = recurseb(num1); // recursive call
        if (borrow == true)
        {
                num1->digit += 9;
                return true; //if success, relay back
        }
        if (borrow == false) return false; //if success, relay back
}

//---------------------------------------
//
void oper_mult(node *num1, node *num2, node *num3)
{

        struct node* extranode;
        int power = 0;
        int k;


        num2 = num2->next;
        for(;;)
        {

                if ((num1->next->digit < 99)) //n1 good, n2 good
                {
                        num1 = num1->next;
                        //printf("advance n1 num1 next = %i, num2 next = %i\n", num1->digit, num2->digit);

                        if (num3->next->digit > 99) //add place to num3 if detect head
                        {
                                extranode = malloc(sizeof(node));
                                extranode->digit = 0;
                                extranode->next = num3->next;
                                num3->next = extranode;
                                extranode->prev = num3;
                                extranode->next->prev = extranode;
                                //printf("extranode = %i\n", extranode->digit);
                        }
                        num3 = num3->next;
                }
                else if ((num1->next->digit > 99) && (num2->next->digit < 99))//n1 end, n2 good
                {
                        power++;

                        num2 = num2->next;
```

```
                //printf("advance n2 num1 next = %i, num2 next = %i\n", num1->digit, num2->digit);
                for(;;)
                {
                        num3 = num3->next;
                        if (num3->digit > 99) break;
                }
                for(k = 0;k <= power;++k)
                {
                        num3 = num3->next;
                }
                for(;;)
                {
                        if (num1->prev->digit > 99) break;
                        num1 = num1->prev;
                }
        }
        //else printf("advancement error\n");

        num3->digit += (num1->digit) * (num2->digit);//add digits
        //printf("%i = %i * %i\n", num3->digit, num1->digit, num2->digit);

        if (num3->digit > 9) //check for carry
        {
                if (num3->next->digit > 99) //add place if detect head
                {
                        extranode = malloc(sizeof(node));
                        extranode->digit = 0;
                        extranode->next = num3->next;
                        num3->next = extranode;
                        extranode->prev = num3;
                        extranode->next->prev = extranode;
                }
                num3->next->digit += num3->digit / 10; //put any tens place digit into next place
                num3->digit %= 10; //put mod as ones place digit in current place
        }

        //check for end
        if ((num1->next->digit > 99) && (num2->next->digit > 99)) // n1 end, n2 end
        {
                //printf("break num1 next = %i, num2 next = %i\n", num1->next->digit, num2->next->digit);
                break;
        }
}

for(;;)
{
        num3 = num3->prev;
```

```
                if (num3->digit > 99) break;
        }

}


//---------------------------------
//
void printbigint(node *num3)
{
        char *answer;
        int slot = 0;
        int len = 1;
        int comma = 0;

        //drop any preceding zeros
        for(;;)
        {
                num3 = num3->prev;
                if ((num3->digit > 0) && (num3->digit < 99))
                {
                        num3 = num3->next;
                        break;
                }
        }

        //count length
        for(;;)
        {
                num3 = num3->prev;
                len++;
                if (num3->digit > 99)
                {
                        break;
                }
        }

        //set up commas
        len += (len-3) / 3;

        //allocate string
        answer = malloc( len * sizeof( char ));
        if (answer == NULL) fatal( "Out of memory." );

        //write sign
        if (num3->digit == 100) {answer[0] = '+';}
        if (num3->digit == 101) {answer[0] = '-';}
```

```
        //write null terminator
        len--;
        answer[len] = '\0';
        len--;

        // copy from list to string, include commas
        for(slot = len; slot > 0; --slot)
        {
                num3 = num3->next;
                answer[slot] = num3->digit + 48;
                comma++;
                if ((comma % 3 == 0) && (slot > 1))
                {
                        slot--;
                        answer[slot] = ',';

                }
        }




        printf("answer is %s", answer);
        printf("\n");
}
```