Recursion

Program #1: Factorial

This computes a factorial recursively.

```
#include <stdio.h>

/*
 * compute n! recursively
 */
int fact(int n)
{
      /* error check */
      if (n < 0)
            return(-1);
      /* base case */
      if (n == 0)
            return(1);
      /* recursion */
      return(n * fact(n-1));
}

/*
 * convert string to int with error checking
 * no leading signs or magnitude checking
 */
int cvttoint(char *s)
{
      int n = 0;        /* integer being read */

      /* skip leading white space */
      while(isspace(*s))
      s++;
      /* if it's not a digit, it's not an integer */
      if (!isdigit(*s))
            return(-1);
      /* read in the integer */
      while(isdigit(*s))
            n = n * 10 + *s++ - '0';
      /* if it's ended by a NUL, it's an integer */
      return(*s ? -1 : n);
}

int main(int argc, char *argv[])
{
      int i;      /* counter in a for loop */
      int n;      /* number read in */
      int rv = 0; /* exit status code */

      /*
       * do each arg separately
       */
      for(i = 1; i < argc; i++)
```

```
                if ((n = cvttoint(argv[i])) != -1)
                      printf("%d! = %d\n", n, fact(n));
                else{
                      /* error handler*/
                      rv++;
                      printf("%s: invalid number\n", argv[i]);
                }

          /*
           * bye!
           */
          return(rv);
    }
```

Program #2. Greatest Common Divisor

This computes the GCD recursively.

```
/*
 * gcd -- compute the GCD of pairs of integers
 *
 * History
 * 1.0      Matt Bishop; original program
 * 1.1      Matt Bishop; made it recursive
 */
#include <stdio.h>

/*
 * macros
 */
#define BAD_GCD -1              /* error in arguments -- */
                                /* MUST be non-positive */

/*
 * recursive GCD
 */
int gcdr(int m, int n)
{
      /* base case(s) */
      if (m == 0)
            return(n);
      if (n == 0)
            return(m);
      /* now recurse */
      return(gcd(n, m % n));
}

/*
 * This function returns the greatest common divisor of its arguments
 * Notes: (1) if m = n = 0, the GCD is undefined -- so we return BAD_GCD
 *        (2) if m < 0 or n < 0, then gcd(m,n) > 0; so we can just make
 *            m and n both positive
 *        (3) if m = 0 and n != 0, gcd(m,n) = n (and vice versa)
 */
int gcd(int m, int n)
{
      int rem;          /* remainder for Euclid's algorithm */

      /*
       * special cases
       */
      /* error check -- if both 0, undefined */
      if (m == 0 && n == 0)
            return(BAD_GCD);
      /* make all negatives positive */
      if (m < 0) m = -m;
      if (n < 0) n = -n;

      /*
```

```
         * now apply the recursive algorithm
         */
        return(gcdr(m, n));
}

/*
 * the main routine
 */
void main(void)
{
        int m, n;         /* numbers to take the GCD of */
        int g;            /* the GCD of m and n */
        int c;            /* used to gobble up rest of line */

        /*
         * loop, asking for numbers and printing the GCD
         */
        while(printf("Enter two numbers: "),
                                      scanf("%d %d", &m, &n) != EOF){
                while((c = getchar()) != EOF && c != '\n')
                        ;
                /* print the result -- note that if the input */
                /* is invalid, gcd() simply returns BAD_GCD   */
                printf("The GCD of %d and %d is ", m, n);
                if ((g = gcd(m, n)) == BAD_GCD)
                        printf("undefined.\n");
                else
                        printf("%d.\n", g);
        }

        /*
         * clean up output and exit
         */
        putchar('\n');
        exit(0);
}
```

Program #3. Sorting

This is a very simple recursive sorting program.

```
#include <stdio.h>

/*
 * the array and its size
 */
int list[] = { 13, 82, 0, 16, 5, -1, 99, 0 };
int nlist = sizeof(list)/sizeof(int);

/*
 * recursive sort -- put smallest element at head of array
 * and then sort the rest
 */
void sort(int l[], int lsz)
{
      int i;          /* counter in a for loop */
      int tmp;        /* used to swap ints */
      int min;        /* index of minimum element */

      /* base case */
      if (lsz == 1)
      return;

      /* find index of smallest number in array */
      min = 0;
      for(i = 1; i < lsz; i++)
      if (l[i] < l[min])
            min = i;

      /* move smallest element to 0-th element */
      tmp = l[0];
      l[0] = l[min];
      l[min] = tmp;

      /* recurse */
      sort(&l[1], lsz-1);
}

int main(void)
{
      int i;            /* counter in a for loop */

      /* print initial array */
      printf("initial array: ");
      for(i = 0; i < nlist;i++)
            printf(" %3d", list[i]);
      putchar('\n');

      /* now sort */
      sort(list, nlist);

      /* print sorted array */
```

```
        printf("final array:   ");
        for(i = 0; i < nlist;i++)
              printf(" %3d", list[i]);
        putchar('\n');
        return(0);
  }
```