```
/*
 * This is a free program sample that may be reproduced in any form.
 * The author's information should be retained to preserve its identity.
 *
 * Date written: Nov 7, 2004
 * Written by:   Peraphon Sophatsathit
 * Department of Mathematics, Faculty of Science, Chulalongkorn University.
 * email: Peraphon.S@chula.ac.th
 * http://pioneer.netserv.chula.ac.th/~sperapho
 *
 * Computer Systems (2301274) class supplement.
 * Description:  This sample program demonstrates how to carry out
 *        simple computations in different number bases (2-16).
 *        The computation consists of ADD and SUB operations
 *        only.  The latter is done by 2's complement
 *        addition, whereby reducing the effective operation
 *        to only ADD.
 * input format: num#base, num#base, command#display_base
 *        e.g.,  234#10, 45#7, add#9
 *            54#8, 6A#13, sub#16
 * error report: For simplicity sake, error conditions are confined
 *        to the following categories:
 *        1. wrong argument  (Warg)
 *            54#8, 6D#12, sub#16
 *            5f#8, 64#8, sub#6
 *        2. missing command (Mcmd)
 *            54#8, 6A#13
 *            54#8, sub#16
 *        3. wrong command   (Wcmd)
 *            54#8, 6A#13, mul#16
 */
#include      <stdio.h>
#include      <stdlib.h>
#include      <string.h>
#include      <ctype.h>

#define       Bsiz          100
#define       Nul           '\0'
#define       delim         ", \t\n"
#define       Base          '#'
#define       Term          3

#define       EQ(a, b)      (strcmp(a, b) == 0)
#define       Zero_chr      '0'
#define       Prompt        "Enter->"

char   dit[] = { "0123456789abcdef" };
```

```
/*
 * the command types, command lists, and error messages
 * must be kept in parallel.  This setup permits
 * new commands to be added or old ones deleted with
 * relatively minor change to the source code.
 */
enum   cmd_types
{
       A_cmd, S_cmd, Normal,
       Warg, Mcmd, Wcmd, Illegal
};
typedef      enum   cmd_types    code;
char   *cmd_list[] =
{
       "add", "sub", ""
};
char   *err_msg[]  =
{
       "", "", "Normal Termination",
       "Wrong argument",
       "Missing command",
       "Wrong command",
       "Illegal command"
};

/*
 * function prototype
 */
int    input_line(char *, int);
code   parse_input(char *);
code   check_cmd(char *);
void   breakup(char *, char *, char *);
void   processing(code);
void   error_report(code);
code   convert(char *, char *, int *);
void   revert(int, char *);

int    first, next, r_base;

/*
 * main processing loop.  Read input from keyboard and
 * parse it into separate tokens for subsequent computations.
 */
int
main(void)
{
       char   buff[Bsiz];
       code   re_type;

       while (input_line(buff, Bsiz-1) > 0)
       {
              if ((re_type = parse_input(buff)) < Normal)
                     processing(re_type);
              else
                     error_report(re_type);
       }
       return 0;
}
```

```
/*
 * read at most N (Bsiz-1) characters from keyboard and
 * get rid of the newline character.
 */
int
input_line(char *s, int N)
{
        int     n, rtcode;

        printf("\n%s", Prompt);
        if (fgets(s, N, stdin) != NULL)
        {
                n = strlen(s);
                rtcode    = n - 1;
                s[rtcode] = Nul;
        }
        else
                rtcode = 0;
        return rtcode;
}

/*
 * parse input buffer based on a set of predetermined
 * delimiters, namely, comma, blank, tab, and newline.
 */
code
parse_input(char *s)
{
        char    token[Bsiz/2];
        char    num[Bsiz/2], bases[Bsiz/2];
        char    *ptr, *p;
        int     tally = 0;
        code    rt;

        strcpy(token, (const char *)strtok_r(s, (const char *)delim, &ptr));
        breakup(token, num, bases);
        tally++;
        rt = convert(num, bases, &first);
        if (rt > Normal)
                return Warg;
        while ((p = (char *)strtok_r(NULL, (const char *)delim, &ptr)) != NULL)
        {
                strcpy(token, p);
                breakup(token, num, bases);
                tally++;
                if (tally == Term - 1)
                {
                        rt = convert(num, bases, &next);
                        if (rt > Normal)
                                return Warg;
                }
        }
        /*
         * check if command is missing
         */
        if (tally != Term)
                rt = Mcmd;
        else
        {
                r_base = atoi(bases);
                rt = check_cmd(num);
        }
        return rt;
}
```

```
/*
 * break up each token into number and radix based on
 * '#' delimiter.
 */
void
breakup(char *token, char *num, char *bases)
{
       char   *t;
       int    len;

       t   = strchr(token, Base);
       len = t - token;
       strncpy(num, token, len);
       num[len]   = Nul;
       len = token + strlen(token) – 1 - t;
       strncpy(bases, t+1, len);
       bases[len] = Nul;
}

/*
 * check if the command is legal by converting all characters
 * to lowercase.
 */
code
check_cmd(char *cmd)
{
       static int    n_cmd = sizeof(cmd_list) / sizeof(char *);
       int    i, c;
       code   rt = Normal;

       for (i = 0; i < strlen(cmd); i++)
       {
              c = cmd[i];
              if (isalpha(c) != 0)
                     cmd[i] = tolower(c);
              else
                     return Wcmd;
       }
       for (i = 0; i < n_cmd; i++)
       {
              if (EQ(cmd, cmd_list[i]))
              {
                     rt = (code)i;
                     break;
              }
       }
       if ((code)i >= Normal)
              rt = Wcmd;
       return rt;
}

/*
 * perform addition and complement addition (2's).
 * A note on programming style: the symbolic constants
 * below are used specifically in 'processing' function,
 * hence they are placed as close to the point of
 * application as possible.
 */
```

```
#define      Neg          "negative"
#define      Nul_str      ""
#define      Small        20

void
processing(code     cmd)
{
       int    out;
       char   out_buff[Bsiz/2], sign[Small];

       strcpy(sign, Nul_str);
       switch (cmd)
       {
              case S_cmd:
                     next = ~next + 1;       /* 1's complement plus one    */
                     out  = first + next;
                     if (abs(next) > abs(first)) /* output is complemented */
                     {
                            out = ~out + 1;
                            strcpy(sign, Neg);
                     }
                     break;
              case A_cmd:
                     out  = first + next;
                     break;
              default:                        /* trap any illegal commands */
                     break;
       };
       revert(out, out_buff);
       printf("\nThe output is %s %s\n", sign, out_buff);
}

/*
 * print the corresponding error messages.
 */
void
error_report(code et)
{
       code   i;
       int    ix;

       for (i = Normal+1; i < Illegal; i++)
              if (i == et)
              {
                     ix = (int)i;
                     printf("\nERROR: %s\n", err_msg[ix]);
                     break;
              }
}


/*
 * convert all inputs to base 10 numbers for internal use.
 */
code
convert(char *num, char *bases, int *out)
{
       code   found   = Illegal;
       int    dit_len = strlen(dit);
       int    size, m, i, j, c;
       int    result, n, val;
```

```
        size = strlen(num);
        m    = atoi(bases);
        if (m > 10)
        {
                for (i = 0; i < size; i++)
                {
                        c = num[i];
                        if (isalpha(c) != 0)
                                num[i] = tolower(c);
                }
        }
        for (result = j = 0; j < size; j++)
        {
                for (i = 0; i < dit_len; i++)
                {
                        if (num[j] == dit[i] && i <= m-1)
                        {
                                found = Normal;
                                break;
                        }
                }
                if (found > Normal)
                        return found;
                for (val = 1, n = 0; n < size-j-1; n++)
                        val *= m;
                val     *= i;
                result  += val;
        }
        *out = result;
        return found;
}

/*
 * convert the internal form to the designated output base.
 */
void
revert(int out, char *s)
{
        int    m, i, radix = 1;
        char   rsum[Bsiz/2];

        i = 0;
        while (out > 0)
        {
                m       = out % r_base;
                out    /= r_base;
                rsum[i] = Zero_chr + m;
                radix  *= r_base;
                i++;
        }
        rsum[i] = Nul;
        i--;
        for (m  = 0; m < strlen(rsum); m++, i--)
                s[m] = rsum[i];
        s[m]    = Nul;
}
```