```
/*
 * 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: October 26, 2005
 * 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
 *
 * Objective:  This is a demonstration program to activate available
 *             functions via an array of function pointers.  It is
 *             ideal for writing C code in object-like fashion.
 * Description:      read two integers and print their sum; repeat till both
 *             input integers are zero simultaneously.
 *             The available functions are purposedly created to incorporate
 *             bugs, thereby mimicking real world scenarios.
 * Test data:  Equivalent partitioning technique is employed to catch
 *             possible errors in each algorithm, namely, for any input
 *             integers n and m,
 *             1. both positive
 *             2. both negative
 *             3. both zero (termination condition)
 *             4. opposite sign and abs(n) = abs(m)
 *             5. opposite sign but abs(n) != abs(m)
 *             6. one zero and the other is not zero
 */

#include      <stdio.h>
#include      <stdlib.h>

#define              First         0      /* first method      */

/*
 * function and algorithm prototypes
 */
void          proc_loop(int);
void          pro1(int *);
void          pro2(int);
void          pro3(void);

/*
 * use 'enum' to designate the available methods, and delimited by 'Last'.
 */
enum   proc_name
{
       P1, P2, P3, Last
};
```

```
typedef       void   (*func_T)(void);    /* generic function type (arbitrary) */
struct func
{
    char     *name;
    func_T   fn;
}
Methods[] =
{
    { "pro1", (func_T)pro1  },
    { "pro2", (func_T)pro2  },
    { "pro3", (func_T)pro3  }
};

/*
 * This is another way to count the number of function methods.
 * The 'enum' approach is a prefer one, however.

int    func_count = sizeof(Methods) / sizeof(struct func) - 1;

 */

int
main(int ac, char *av[])
{
       int    style;

       switch (ac)
       {
              case 2:
                     style = atoi(av[1]);
                     proc_loop(style);
                     break;
              default:
                     printf("Usage: %s [desired algorithm: %d-%d]\n", av[0], First, Last-1);
       }
       return 0;
}

/*
 * main loop to invoke the desired algorithm specified from command line
 */
void
proc_loop(int sty)
{
       int    val;


       if (sty < First || sty >= Last)
       {
              printf("out of range\n\n");
              return;
       }

       /*
        * could be done both ways
        * first approach: harder to interpret, hence commented out.
        *

       (*(void *(*)(void))Methods[sty].fn)();

        */
```

```
	/*
	 * or second approach: easier
	 *

	(*Methods[sty].fn)();

	 */


	/*
	 * in case each function returns value other than 'void', hence cast
	 * to 'val' (int).  In this case, the cast is useless since the functions
	 * return void.
	 */
	switch (sty)
	{
		case P1:
			val = (*(int (*)(int *))Methods[sty].fn)(&sty);
			break;
		case P2:
			val = (*(int (*)(int))Methods[sty].fn)(sty);
			break;
		case P3:
			/*
			 * could be invoked both ways

			val = (*(int (*)(void))Methods[sty].fn)();

			 */

			(*Methods[sty].fn)();
			break;
		default:
			break;
	}
}

/*
 * correct solution
 */
void
pro1(int *x)
{
	int	a, b;

	printf("pro1: parm = %d\n", *x++);
	printf("enter two integers (0 0 to quit) ");
	(void)scanf("%d%d", &a, &b);
	while (a != 0 || b != 0)
	{
		printf("the sum is: %d\n\n", a+b);
		printf("enter two integers (0 0 to quit) ");
		(void)scanf("%d%d", &a, &b);
	}
}
```

```
/*
 * reverse (wrong) logic: ||  becomes  &&
 * fails when either input is zero
 */
void
pro2(int x)
{
	int	a, b;

	printf("pro2: parm = %d\n", x++);
	printf("enter two integers (0 0 to quit) ");
	(void)scanf("%d%d", &a, &b);
	while (a != 0 && b != 0)
	{
		printf("the sum is: %d\n\n", a+b);
		printf("enter two integers (0 0 to quit) ");
		(void)scanf("%d%d", &a, &b);
	}
}

/*
 * reverse comparison: !=  becomes  ==
 * fails when both inputs are non-zero (simultaneously)
 */
void
pro3(void)
{
	int	a, b;

	printf("pro3: no parm\n");
	printf("enter two integers (0 0 to quit) ");
	(void)scanf("%d%d", &a, &b);
	while (a == 0 || b == 0)
	{
		printf("the sum is: %d\n\n", a+b);
		printf("enter two integers (0 0 to quit) ");
		(void)scanf("%d%d", &a, &b);
	}
}
```