```
/*---------------------------------------------------------------------*/
/* ptrs.c                                                              */
/* Author: Bob Dondero                                                 */
/*---------------------------------------------------------------------*/

#include <stdio.h>

/*---------------------------------------------------------------------*/

/* Illustrate pointers. Return 0. */

int main(void)
{
   /*------------------------------------------------------------------*/
   /* Pointer Fundamentals                                             */
   /*------------------------------------------------------------------*/

   int i1;      /* i1 is a variable of type int. */

   int i2;      /* i2 is a variable of type int. */

   int *pi3;    /* pi3 is a variable of type int*. */
                /* pi3 is an integer pointer. */

   int* pi4;    /* pi4 is a variable of type int*. */
                /* Spacing before and after "*" doesn’t matter. */

   i1 = 5;

   /* pi4 = 6;     Compiletime warning: type mismatch. */
   /* pi3 = i1;    Compiletime warning: type mismatch. */
   /* i1 = pi3;    Compiletime warning: type mismatch. */

   pi3 = &i1;   /* "&" is the "address of" operator. */

   /* pi3 = 6;     Still compiletime warning. */

   *pi3 = 6;    /* "*" is the "dereference" operator. */
                /* Changes value of *pi3 and i1. */
                /* *pi3 and i1 are aliases. */
                /* Here: undesirable.  Elsewhere: useful. */

   /* *pi4 = 7;    Runtime error. Seg fault, or memory corruption. */

   pi4 = &i2;   /* Hereafter, *pi4 and i2 are aliases. */

   i2 = *pi3;   /* Assigns an int to an int variable. */

   *pi4 = *pi3; /* Same as previous. */

   pi4 = pi3;   /* Assigns an address to a pointer. */
                /* *pi3 and *pi4 are now aliases. */

   pi4 = &i2;   /* Restore pi4 to previous value */

   /* & and * are inverse operators:
         If you write &*pi3, then you might as well write pi3 instead.
         If you write *&i1, then you might as well write i1 instead. */

   /*------------------------------------------------------------------*/
   /* The NULL address                                                 */
   /*------------------------------------------------------------------*/

```

ptrs.c (Page 2 of 2)

```
   pi3 = NULL;  /* Indicates that pi3 points to no valid memory
                   location. */

   /* NULL is a #defined constant in several standard header files. */
   /* #define NULL (void*)0 */

   /* NULL differs from "unpredictable value." */
   /* *pi3 = 8;    Runtime error: Seg fault. */

   pi3 = &i1;   /* Restore value of pi3 */

   /*-----------------------------------------------------------------*/
   /* Pointers and Relational Operators                               */
   /*-----------------------------------------------------------------*/

   if (*pi3 == *pi4)  /* Compares ints. Evaluates to TRUE (1). */
      printf("Integers are equal\n");

   if (pi3 == pi4)    /* Compares addresses. Evaluates to FALSE (0). */
      printf("Pointers are equal\n");

   if (pi3 != pi4)    /* Compares addresses. Evaluates to TRUE (1). */
      printf("Pointers are unequal\n");

   /* Note:
      if (pi3 == pi4) is TRUE,
         then (*pi3 == *pi4) is TRUE.
      if (*pi3 == *pi4) is TRUE,
         then (pi3 == pi4) may or may not be TRUE.  */

   if (pi3 == NULL)   /* Compares addresses. Evaluates to FALSE (0). */
      printf("Pointer is NULL\n");

   return 0;
}
```

testquorembad.c (Page 1 of 1)

```
/*--------------------------------------------------------------------*/
/* testquorembad.c                                                    */
/* Author: Bob Dondero                                                */
/*--------------------------------------------------------------------*/

#include <stdio.h>
#include <assert.h>

/*--------------------------------------------------------------------*/

/* Divide iDividend by iDivisor.  Assign the remainder to iRemainder,
   and return the quotient. */

static int quorem(int iDividend, int iDivisor, int iRemainder)
{
   assert(iDivisor != 0);
   iRemainder = iDividend % iDivisor;
   return iDividend / iDivisor;
}

/*--------------------------------------------------------------------*/

/* Test the quorem function.  Return 0. */

int main(void)
{
   int iQuo;
   int iRem;

   iQuo = quorem(11, 3, iRem);
   printf("Quotient: %d  Remainder: %d\n", iQuo, iRem);

   return 0;
}

/*--------------------------------------------------------------------*/

/* Sample Execution:

$ ./testquorembad
Quotient: 3  Remainder: 8299968

*/
```

testquorem.c (Page 1 of 1)

```c
/*--------------------------------------------------------------------*/
/* testquorem.c                                                       */
/* Author: Bob Dondero                                                */
/*--------------------------------------------------------------------*/

#include <stdio.h>
#include <assert.h>

/*--------------------------------------------------------------------*/

/* Divide iDividend by iDivisor.  Assign the remainder to
   *piRemainder, and return the quotient. */

static int quorem(int iDividend, int iDivisor, int *piRemainder)
{
   assert(iDivisor != 0);
   assert(piRemainder != NULL);
   *piRemainder = iDividend % iDivisor;
   return iDividend / iDivisor;
}

/*--------------------------------------------------------------------*/

/* Test the quorem function.  Return 0. */

int main(void)
{
   int iQuo;
   int iRem;

   iQuo = quorem(11, 3, &iRem);
   printf("Quotient: %d  Remainder: %d\n", iQuo, iRem);

   return 0;
}

/*--------------------------------------------------------------------*/

/* Sample Execution:

$ ./testquorem
Quotient: 3  Remainder: 2

*/
```

testswapbad.c (Page 1 of 1)

```c
/*--------------------------------------------------------------------*/
/* testswapbad.c                                                      */
/* Author: Bob Dondero                                                */
/*--------------------------------------------------------------------*/

#include <stdio.h>

/*--------------------------------------------------------------------*/

/* Swap the values of iFirst and iSecond. */

static void swap(int iFirst, int iSecond)
{
   int iTemp;

   iTemp = iFirst;
   iFirst = iSecond;
   iSecond = iTemp;
}

/*--------------------------------------------------------------------*/

/* Test the swap function. Return 0. */

int main(void)
{
   int i1 = 8;
   int i2 = 12;

   printf("Before:  %d %d\n", i1, i2);

   swap(i1, i2);

   printf("After:  %d %d\n", i1, i2);

   return 0;
}

/*--------------------------------------------------------------------*/

/* Sample Execution:

$ ./testswapbad
Before:  8 12
After:  8 12

*/
```

testswap.c (Page 1 of 1)

```c
/*--------------------------------------------------------------------*/
/* testswap.c                                                         */
/* Author: Bob Dondero                                                */
/*--------------------------------------------------------------------*/

#include <stdio.h>
#include <assert.h>

/*--------------------------------------------------------------------*/

/* Swap the values of *piFirst and *piSecond. */

static void swap(int *piFirst, int *piSecond)
{
   int iTemp;

   assert(piFirst != NULL);
   assert(piSecond != NULL);

   iTemp = *piFirst;
   *piFirst = *piSecond;
   *piSecond = iTemp;
}

/*--------------------------------------------------------------------*/

/* Test the swap function.  Return 0. */

int main(void)
{
   int i1 = 8;
   int i2 = 12;

   printf("Before:  %d %d\n", i1, i2);

   swap(&i1, &i2);

   printf("After:  %d %d\n", i1, i2);

   return 0;
}

/*--------------------------------------------------------------------*/

/* Sample Execution:

$ ./testswap
Before:  8 12
After:  12 8

*/
```

Princeton University
COS 217: Introduction to Programming Systems
Kinds of Function Parameters

Kind of Parameter	Example	Implementation	C Construct
in	`IntMath_gcd()` (both params)	call by value	ordinary parameter
out	`quorem()` (3rd param) `scanf()` (2nd param)	call by reference	pointer parameter
inout	`swap()` (both params)	call by reference	pointer parameter

rev.c (Page 1 of 1)

```c
/*--------------------------------------------------------------------*/
/* rev.c                                                              */
/* Author: Bob Dondero                                                */
/*--------------------------------------------------------------------*/

#include <stdio.h>

/*--------------------------------------------------------------------*/

/* Read ARRAY_LENGTH integers from stdin, and write them in reverse
   order to stdout.  Return 0. */

int main(void)
{
   enum {ARRAY_LENGTH = 5};

   int aiNums[ARRAY_LENGTH];
   int i;

   printf("Enter %d integers:\n", ARRAY_LENGTH);
   for (i = 0; i < ARRAY_LENGTH; i++)
      scanf("%d", &aiNums[i]);

   printf("\n");

   printf("The integers in reverse order are:\n");
   for (i = ARRAY_LENGTH-1; i >= 0; i--)
      printf("%d\n", aiNums[i]);

   return 0;
}
```

Princeton University
COS 217: Introduction to Programming Systems
Pointer-Related Operators

Key

```
p, p1, p2     Pointer variables
i             An integral expression
```

Operators Meaningful for Any Pointer Variable

Dereference Operator

```
*p            The contents of the memory referenced by p.
```

Equality and Inequality Relational Operators

```
p1 == p2      1 if p1 is equal to p2, and 0 otherwise.
p1 != p2      1 if p1 is unequal to p2, and 0 otherwise.
```

Assignment Operator

```
p1 = p2       Side effect:  Assign p2 to p1.  The new value of p1.
```

Operators Meaningful for Pointers that Reference Array Elements

Arithmetic Operators

```
p + i         The address of the ith element after the one referenced by p.
i + p         The address of the ith element after the one referenced by p.
p – i         The address of the ith element before the one referenced by p.
p++           Side effect:  Increment p to point to the next element.
              The previous value of p.
++p           Side effect:  Increment p to point to the next element.
              The new value of p.
p--                  Side effect:  Decrement p to point to the previous element.
              The previous value of p.
--p                  Side effect:  Decrement p to point to the previous element.
              The new value of p.
```

Arithmetic Operators

```
p1 - p2       The "span" of p1 and p2.
```

Relational Operators

```
p1 < p2       1 if p1 is less than p2, and 0 otherwise.
p1 <= p2      1 if p1 is less than or equal to p2, and 0 otherwise.
p1 > p2       1 if p1 is greater than p2, and 0 otherwise.
p1 >= p2      1 if p1 is greater than or equal to p2, and 0 otherwise.
```

Assignment Operators

```
p += i        Side effect:  Increment p so its value is the address of
              the ith element after the one referenced by p.
              The new value of p.
p -= i        Side effect:  Decrement p so its value is the address of
              the ith element before the one referenced by p.
              The new value of p.
```

Disallowed

```
p1 + p2
i - p
i += p
i -= p
p == i
```

Array Subscripting Operator

```
p[i]          *(p + i), that is, the contents of memory at the address
              that is i elements after the address referenced by p.
```