Pointer_temperature.c (not annotated)

```
#include <stdio.h>
void convert(double *celsius);
int main(){
      double temperature = 80;
      printf("\n\t80 degrees fahrenheit (before conversion) == %f
celsius",temperature);
      convert(&temperature);
      printf("\n\t80 degrees fahrenheit (after conversion) == %f
celsius\n\n",temperature);
return(0);
}
void convert(double *celsius){
      double temp;
      temp = *celsius;
      temp = (temp - 32)/1.8;
      *celsius = temp;
}
```

Pointer_temperature.c (annotated)

```
/*
Illustrates Pointers
   */
#include <stdio.h>
/* Function Protype Statement for the convert function called from
   main() -- "double *celsius" means that a memory address of a double
   variable is going to be passed to the function -- the word
   "celsius" is just a convenience for the programmer to remember what
   the function is going to do!! */
/* Note that the statement
double *celsius;
means that "celsius" is a pointer of type double.  This does not
initialize "celsius".  To do that would require a statement like:

celsius = &heat;

This would result in the pointer "celsius" pointing to the memory
location of the variable "heat".
*/
/* The "void" means that the function convert() does not return any value*/
/* This is the function prototype */
void convert(double *celsius);

int main(void){
        double temperature = 80;
/* the \t is a Tab */
        printf("\n\t80 degrees fahrenheit (before conversion) == %f
celsius",temperature);
/* Pointer arguments enable a function to access and change objects in
 * the function that calls it -- In this case, temperature in main()
 * above gets changed in the convert() function by using a pointer to
 * the memory location of temperature in main() */
/* "&temperature" means "memory address of the variable temperature"*/
        convert(&temperature);
        printf("\n\t80 degrees fahrenheit (after conversion) == %f
celsius\n\n",temperature);
return(0);
}
void convert(double *celsius){
        double temp;
/* temp is set to the value pointed to by "*celsius" -- that is, "80" */
        temp = *celsius;
        temp = (temp - 32)/1.8;
/* Applying the "*" operator to a pointer -- in this case "celsius" is a
 * pointer -- yields the value stored in
 * the pointed-to object.  Here "celsius" is a pointer to the memory
 * location of temperature in the main program -- the value "80".
 * "*celsius" replaces "80" with the value in temp, or "26.6666" at the
 * memory location &temperature */
        *celsius = temp;
/* Note that there is no "return" used because the function is declared
 * as a "void" */
}
```

Function_Example.c

```
/* Program illustrates two types of function calls -- one a simple math
 * function the other uses pointers to swap two values of variables in
 * memory */
/* Program also illustrates how the clock() function can be used to
 * time portions of the program. */
#include <time.h>
#include <math.h>
#include <stdlib.h>
#include <stdio.h>
/* Here are the two function prototypes that we use below -- Note that
 * the first is passing the memory addresses of two double numbers while
 * the second passes variable values */
void interchange(double * x, double * y);
double power_double(double x, double y);
FILE *kp;

int main(void)
{
/* use doubles to show small values of time */
         double time1, time2, time3, timedif;
         int i;
         double temp[10000];
         double temp2, temp3;
/* clock() is part of time.h -- returns the implementation's
 * best approximationto the processor time elapsed since the
 * program was invoke, divide by CLOCKS_PER_SEC to get the time
 * in seconds          */
         time1 = (double) clock();            /* get initial time */
         time1 = time1 / CLOCKS_PER_SEC;      /*    in seconds    */
         kp=fopen("function_example.dat","w");
         srand(17);
/* check operating system limit value */
         printf("RAND_MAX=%10d\n", RAND_MAX);
         for(i=0;i<10000;i++){
/* convert RAND_MAX into double*/
                  temp[i] = rand()/((double)RAND_MAX + 1);
                  printf("rand = %10.6f\n",temp[i]);
                  fprintf(kp,"rand = %10.6f\n",temp[i]);
               printf("i = %d\n",i);
         }

         timedif = ( ((double) clock()) / CLOCKS_PER_SEC) - time1;
         printf("The elapsed time is %12.8f seconds\n", timedif);
         time2 = (double) clock();
         time2 = time2 / CLOCKS_PER_SEC;
         for(i=0;i<10000;i++){
                  temp2 = rand()/((double)RAND_MAX + 1);
                  temp3 = power_double(temp[i],temp2);
                  fprintf(kp,"power_double %10.6f, %10.6f, %10.6f\n",temp[i],temp2,temp3);
         }
         timedif = ( ((double) clock()) / CLOCKS_PER_SEC) - time2;
         printf("The elapsed time is %12.8f seconds\n", timedif);

         time3 = (double) clock();
         time3 = time3 / CLOCKS_PER_SEC;
         for(i=0;i<10000;i++){
                  temp2 = rand()/((double)RAND_MAX + 1);
/* Note that two memory addresses are passed to the interchange function */
                  interchange(&temp[i],&temp2);
                  temp3 = power_double(temp[i],temp2);
                  fprintf(kp,"power_double_interchange %10.6f, %10.6f,
%10.6f\n",temp[i],temp2,temp3);
         }
         timedif = ( ((double) clock()) / CLOCKS_PER_SEC) - time3;
         printf("The elapsed time is %12.8f seconds\n", timedif);
         timedif = ( ((double) clock()) / CLOCKS_PER_SEC) - time1;
```

```
          printf("The total elapsed time of the program is %12.8f seconds\n", timedif);
          fclose(kp);
          return(0);
}
double power_double(double x, double y){
          double sum;
          sum = pow(x,y);
          return sum;
}
void interchange(double * x, double * y){
          double xchanger;
/* Put the value of the variable at x's memory location into xchanger – Note dereferencing*/
          xchanger = *x;
/* Put the value of the variable at y's memory location into x's memory location – dereferencing
*/
          *x = *y;
/* Put the value that was at x's memory location into y's memory location -- dereferencing*/
          *y = xchanger;
}
```

writedata.c

```
/* writedata.c -- Writes out two files using the rand() function.  One
 * file is a simple column of data and the second file is a matrix of
 * data*/
#include <stdlib.h>
#include <stdio.h>
/* Declare pointers to the two files */
FILE *fp;
FILE *kp;

int main(void){

    double data[10];
    int i = 0;
    int j = 0;
    srand(14);
/* Open the two files */
    fp = fopen("data.txt","w");
    kp = fopen("data2.txt","w");
    for(i=0;i<10;i++){
        data[i] = ( (double)rand() / ((double)(RAND_MAX)+(double)(1)));
     }

    for(i=0;i<10;i++){
/*    fprintf(fp, "%7.3f\n", data[i]); */
    fprintf(fp, "%f\n", data[i]);
    }
/* For loops for writing out data[.] in 100 rows of 10 numbers per
     * row */
    for(j=0;j<100;j++)
    {
            for(i=0;i<10;i++)
            {
                    data[i] = ( (double)rand() / ((double)(RAND_MAX)+1));
/*                  fprintf(kp, "%7.3f",data[i]); */
                    fprintf(kp, "%f",data[i]);
            }
/* This is the line feed -- newline at the end of the the jth row */
            fprintf(kp,"\n");
    }
    fclose(fp);
    fclose(kp);
 return(0);
}
```

readdata.c (not annotated)

```
/*
gcc -Wall %1.c -o %1.exe
   */
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

int main(void) {

    int MAXNUMBERS = 10000;
    FILE *fp;
 /* numlist is a pointer of type double */
    double *numlist;
    char llist;
    numlist = (double *) malloc (MAXNUMBERS*sizeof(double));
    int i;

    if((fp = fopen("c:/docs_c_summer_course/data.txt","r"))==NULL)
    {
        printf("\nUnable to open file DATA.TXT: %s\n", strerror(errno));
       exit(EXIT_FAILURE);
    }
    else {
           i=0;
        while (!feof(fp)) {
               fscanf(fp,"%c%lf", &llist, &numlist[i]);
               printf("%7.3f\n", numlist[i]);
          i++;
        }
    }

    fclose(fp);
    numlist = (double *) realloc(numlist,  i* sizeof(double));
    printf("\nAllocation OK, %i vector entries allocated.\n", i);
    free(numlist);
    return(0);
}
```

readdata.c (annotated)

```
/*
gcc -Wall %1.c -o %1.exe
  0.003
  0.828
  0.281
  0.355
  0.677
  0.038
  0.585
  0.735
  0.373
  0.307
   */
#include <stdlib.h>
#include <stdio.h>
/* The string library includes the strerror() function*/
#include <string.h>

int main(void) {
    int MAXNUMBERS = 10000;
    FILE *fp;
 /* numlist is a pointer of type double */
    double *numlist;
/* Allocate memory 10000 doubles -- numlist points to the start of this
 * memory block*/
    numlist = (double *) malloc (MAXNUMBERS*sizeof(double));
    int i;
    if((fp = fopen("c:/docs_c_summer_course/data.txt","r"))==NULL)
    {
/* function strerror(errno) returns a pointer to an
 * implementation-dependent error message string corresponding to the
 * error number stored in errno */
        printf("\nUnable to open file DATA.TXT: %s\n", strerror(errno));
/* Part of the stdlib.h library -- abnormal termination of program --
 * the equivalent is to use exit(1);  The prototype is void exit(int
 * status) */
        exit(EXIT_FAILURE);
    }
    else {
           i=0;
/* Part of stdio.h library -- Tests for end of file --
   * "!feof(fp)" means "keep looping until you hit the end of
   * the file*/
        while (!feof(fp)) {
/* “%lf” means “long float”, that is, a double!! */
              fscanf(fp,"%lf", &numlist[i]);
              printf("%7.3f\n", numlist[i]);
          i++;
        }
    }
    fclose(fp);
/* realloc changes the size of the block of memory pointed to by the
 * pointer "numlist" */
```

```
    numlist = (double *) realloc(numlist,  i* sizeof(double));
    printf("\nAllocation OK, %i vector entries allocated.\n", i);
/* frees the space pointed by the numlist pointer -- this is good
 * programming practice */
    free(numlist);
    return(0);
}
```

Readdata_eof.c (annotated)

```
/*
gcc -Wall %1.c -o %1.exe
  0.003
  0.828
  0.281
  0.355
  0.677
  0.038
  0.585
  0.735
  0.373
  0.307
   */
#include <stdlib.h>
#include <stdio.h>
/* The string library includes the strerror() function*/
#include <string.h>

int main(void) {

    int MAXNUMBERS = 10000;
    FILE *fp;
/* numlist is a pointer of type double */
    double *numlist;
/* Allocate memory 10000 doubles -- numlist points to the start of this
 * memory block*/
    numlist = (double *) malloc (MAXNUMBERS*sizeof(double));
    int i, k, kk, imax;

    if((fp = fopen("c:/docs_c_summer_course/data.txt","r"))==NULL)
    {
/* function strerror(errno) returns a pointer to an
 * implementation-dependent error message string corresponding to the
 * error number stored in errno */
        printf("\nUnable to open file DATA.TXT: %s\n", strerror(errno));
/* Part of the stdlib.h library -- abnormal termination of program --
 * the equivalent is to use exit(1);  The prototype is void exit(int
 * status) */
        exit(EXIT_FAILURE);
    }
    else {
        for(i=0;;i++)
        {
        fscanf(fp,"%lf", &numlist[i]);
/* if the end of the file is not reached feof returns "0".  If the end
 * of the file is encountered then feof returns an integer that is not
 * zero. */
        kk=feof(fp);
        printf(" i = %d, EOF = %d\n",i,kk);
        if(kk != 0)
                {
                i = i - 1;
/* break is a keyword that causes program control to skip the rest of
```

```
 * the loop and to resume with the next command following the loop */
                break;
                 }
        }
    }
    fclose(fp);
/* k is the number of ENTRIES in numlist[.] -- the counter "i" runs
 * from 0,1,2,3,4,5,6,7,8,9 -- so we subtract "1" above and set it
 * equal to "imax in case we want to run another loop from 0 to imax */
    imax = i;
    k = i + 1;
/* realloc changes the size of the block of memory pointed to by the
 * pointer "numlist" */
    numlist = (double *) realloc(numlist,  k* sizeof(double));
    printf("\nAllocation OK, %d vector entries allocated.\n", k);
/* frees the space pointed by the numlist pointer -- this is good
 * programming practice */
    free(numlist);
    return(0);
}
```

Readdata_matrix.c (annotated)

```
/*
gcc -Wall %1.c -o %1.exe
   */
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

int main(void) {

    int MAXNUMBERS = 10000;
    FILE *fp;
 /* numlist is a pointer of type double */
    double *data;
/* Allocate memory 10000 doubles -- numlist points to the start of this
 * memory block*/
    data = (double *) malloc (MAXNUMBERS*sizeof(double));
    int i, j, kk;

    if((fp = fopen("c:/docs_c_summer_course/data2.txt","r"))==NULL)
    {
        printf("\nUnable to open file DATA2.TXT: %s\n", strerror(errno));
       exit(EXIT_FAILURE);
    }
    else {
/* Lazy way of reading in the data!!  But it works great if you know what
your data look like */
           kk = 0;
           for(j=0;j<100;j++)
           {
                  for(i=0;i<10;i++)
                  {
                         fscanf(fp, "%lf",&data[kk]);
                         kk = kk + 1;
                  }
           }
           printf(" total read = %d",kk);
    }

    fclose(fp);
    data = (double *) realloc(data,  kk* sizeof(double));
    printf("\nAllocation OK, %i vector entries allocated.\n", kk);

    for(j=0;j<100;j++)
    {
           for(i=0;i<10;i++)
           {
                  printf("%7.3f",data[j*10 + i]);
           }
           printf("\n");
    }
    free(data);
    return(0);
}
```