Supplement To C Programming

By:
Jim Polzin

e-mail: james.polzin@normandale.edu
otto: http://otto.normandale.edu
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>TABLE OF CONTENTS</td>
<td>2</td>
</tr>
<tr>
<td>RANDOM NUMBER GENERATION</td>
<td>3</td>
</tr>
<tr>
<td>A DECK OF CARDS</td>
<td>10</td>
</tr>
<tr>
<td>BASIC MULTIDIMENSIONAL ARRAYS - EXAMPLES</td>
<td>17</td>
</tr>
<tr>
<td>MULTIDIMENSIONAL ARRAYS &amp; POINTERS - EXAMPLES</td>
<td>28</td>
</tr>
<tr>
<td>LINKED LISTS</td>
<td>32</td>
</tr>
</tbody>
</table>
RANDOM NUMBER GENERATION

- C has a function `rand( )` that returns a pseudo-random sequence of integers on successive calls.
- The random sequence can be used to simulate a random set of events.
- The random values can be manipulated to produce sets of various other values randomly. For instance: Rolling a die, flipping a coin, shuffling a deck of cards,… and so on.

- The sequence of random numbers can be changed by “seeding” the random number generator, calling the function `srand( )` before producing numbers with `rand( )`.

Ex:

```c
/*     FILE: Random1.c     */
/* Random number generation.
- the function rand( ), when called repeatedly will produce a pseudo-random sequence of integers in the range of 0 to RAND_MAX */
#include <stdio.h>
#include <stdlib.h>

int main( )
{
    int i, value;
    printf("Random max = %d\n", RAND_MAX);
    for(i=0; i < 10; i++){
        value = rand( );
        printf("Random value = %d\n", value);
    }
    return 0;
}

/*    OUTPUT: Random1.c
    Random max = 32767
    Random value = 41
    Random value = 18467
    Random value = 6334
    Random value = 26500
    Random value = 19169
    Random value = 15724
    Random value = 11478
    Random value = 29358
    Random value = 26962
    Random value = 24464
*/```
Ex:

/*     FILE: Random2.c     */

/* Random number generation.

- The random values produced by
  rand( ) can be altered or
  mapped to simulate some
  random event.

- The following two loops will
  simulate:
  - flipping a coin
  - rolling a six sided die

*/

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

int main( )
{
    int i, value;

    for(i=0; i < 10; i++){
        value = rand( )%2;
        printf("Heads or tails = %d\n", value);
    }

    printf("\n\n");
    for(i=0; i < 10; i++){
        value = rand( )%6;
        printf("Die value = %d\n", value + 1);
    }

    return 0;
}

/*    OUTPUT: Random2.c

  Heads or tails = 1
  Heads or tails = 1
  Heads or tails = 0
  Heads or tails = 0
  Heads or tails = 1
  Heads or tails = 0
  Heads or tails = 0
  Heads or tails = 0
  Heads or tails = 0
  Heads or tails = 0

  Die value = 6
  Die value = 6
  Die value = 2
  Die value = 4
  Die value = 2
  Die value = 6
  Die value = 2
  Die value = 3
  Die value = 4
  Die value = 1

*/
Ex:

/* FILE: Random3.c */

/* Random number generation.

   - Output can be produced to make
     sense to the user of the program. */

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

int main( )
{
    int i, value;
    for(i=0; i < 10; i++){
        value = rand( )%2;
        printf("Heads or tails = %s\n", value?"Tails":"Heads");
    }
    printf("\n\n");
    for(i=0; i < 10; i++){
        value = rand( )%6;
        printf("Die value = %d, value + 1);  
    }
    return 0;
}

/* OUTPUT: Random3.c

   Heads or tails = Tails
   Heads or tails = Tails
   Heads or tails = Heads
   Heads or tails = Heads
   Heads or tails = Tails
   Heads or tails = Heads
   Heads or tails = Heads
   Heads or tails = Heads
   Heads or tails = Heads
   Heads or tails = Heads

   Die value = 6
   Die value = 6
   Die value = 2
   Die value = 4
   Die value = 2
   Die value = 6
   Die value = 2
   Die value = 3
   Die value = 4
   Die value = 1

   */
Ex:

```c
/*     FILE: Random4.c     */
/* Random number generation.                        */
    /* The function srand( ) "seeds" or       */
    /* starts the random number generator */
    /* at a different point.                */
#include <stdio.h>
#include <stdlib.h>

int main( )
{
    int i, value;
    srand(25);
    for(i=0; i < 10; i++){
        value = rand( )%2;
        printf("Heads or tails = %s\n", value?"Tails":"Heads");
    }
    printf("\n\n");
    for(i=0; i < 10; i++){
        value = rand( )%6;
        printf("Die value = %d\n", value + 1);
    }
    return 0;
}

/*    OUTPUT: Random4.c    */
Heads or tails = Heads
Heads or tails = Tails
Heads or tails = Heads
Heads or tails = Tails
Heads or tails = Heads
Heads or tails = Tails
Heads or tails = Heads
Heads or tails = Tails
Heads or tails = Heads
Heads or tails = Heads

Die value = 2
Die value = 1
Die value = 3
Die value = 6
Die value = 4
Die value = 2
Die value = 5
Die value = 6
Die value = 2
Die value = 1

*/```
Ex:

/*     FILE: Random5.c     */
/* Random number generation.
   - Try different seeds.        */

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

int main( )
{
    int i, value;

    printf("Enter an integer: ");
    scanf("%d", &value);

    srand(value);
    for(i=0; i < 10; i++)
    {
        value = rand()%2;
        printf("Heads or tails = %s\n", value?"Tails":"Heads");
    }

    printf("\n\n");
    for(i=0; i < 10; i++)
    {
        value = rand()%6;
        printf("Die value = %d\n", value + 1);
    }

    return 0;
}

/*    OUTPUT: Random5.c    */

Enter an integer: 217
Heads or tails = Tails
Heads or tails = Tails
Heads or tails = Tails
Heads or tails = Heads
Heads or tails = Tails
Heads or tails = Tails
Heads or tails = Tails
Heads or tails = Tails
Heads or tails = Tails

Die value = 4
Die value = 2
Die value = 1
Die value = 3
Die value = 1
Die value = 2
Die value = 1
Die value = 4
Die value = 4

cont…
Enter an integer: 55
Heads or tails = Heads
Heads or tails = Tails
Heads or tails = Tails
Heads or tails = Tails
Heads or tails = Heads
Heads or tails = Heads
Heads or tails = Tails
Heads or tails = Tails
Heads or tails = Heads

Die value = 1
Die value = 2
Die value = 5
Die value = 4
Die value = 5
Die value = 3
Die value = 3
Die value = 1
Die value = 4

Enter an integer: 217
Heads or tails = Tails
Heads or tails = Tails
Heads or tails = Tails
Heads or tails = Heads
Heads or tails = Tails
Heads or tails = Tails
Heads or tails = Tails
Heads or tails = Tails
Heads or tails = Tails

Die value = 4
Die value = 2
Die value = 1
Die value = 1
Die value = 3
Die value = 1
Die value = 2
Die value = 1
Die value = 4
Die value = 4

*/
Ex:
/*     FILE: Random6.c     */

/* Random number generation.
- Seed from the system clock to
  produce an "unpredictable" starting
  point for the random sequence.  */

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

int main( ) {
    int i, value;
    time_t seed;
    seed = time(NULL);
    srand(seed);
    for(i=0; i < 10; i++){
        value = rand()%2;
        printf("Heads or tails = %s\n", value?"Tails":"Heads");
    }
    printf("\n\n");
    for(i=0; i < 10; i++){
        value = rand()%6;
        printf("Die value = %d\n", value + 1);
    }
    return 0;
}

/*    OUTPUT: Random6.c
Heads or tails = Heads
Heads or tails = Heads
Heads or tails = Tails
Heads or tails = Heads
Heads or tails = Tails
Heads or tails = Tails
Heads or tails = Tails
Heads or tails = Tails
Heads or tails = Heads

Die value = 3
Die value = 6
Die value = 1
Die value = 2
Die value = 1
Die value = 1
Die value = 1
Die value = 5
Die value = 1
Die value = 1

*/
A DECK OF CARDS

- A deck of playing cards can be simulated using the numbers 0-51, 52 different values for 52 different cards.
- Each value will need to be translated to a specific playing card.
- Advantages of this particular model for a deck of cards is the minimal amount of data needed to represent each card, and the ease of processing simple integers.

- Examples below show how to translate the 52 integers into 52 different cards

Ex:

```c
/*     FILE: Cards.c     */
/* Simulate a deck of playing cards. 
   - A card deck can be simulated with 
     the values 0 to 51 representing 
     the 52 cards in the deck. */

#include <stdio.h>

int main( )
{
    int i;
    for(i=0; i < 52; i++){
        printf("Card = %d\n", i);
    }
    return 0;
}

/*    OUTPUT: Cards.c 
    Card = 0
    Card = 1
    Card = 2
    Card = 3
    Card = 4
    Card = 5
    Card = 6
    Card = 7
    Card = 8
    Card = 9
    Card = 10
    ...
    Card = 48
    Card = 49
    Card = 50
    Card = 51
*/```
Ex:

```c
/*     FILE: Cards2.c     */
/* Simulate a deck of playing cards.
   - Produce the suit.     */

#include <stdio.h>

int main( )
{
    int i;
    for(i=0; i < 52; i++)
    {
        printf("Card = %2d %2d\n", i, i/13);
    }
    return 0;
}

/*    OUTPUT: Cards2.c    */
Card =  0  0
Card =  1  0
Card =  2  0
Card =  3  0
Card =  4  0
Card =  5  0
Card =  6  0
Card =  7  0
Card =  8  0
Card =  9  0
Card = 10  0
Card = 11  0
Card = 12  0
Card = 13  1
Card = 14  1
Card = 15  1
Card = 16  1
Card = 17  1
Card = 18  1
Card = 19  1
Card = 20  1
Card = 21  1
Card = 22  1
Card = 23  1
Card = 24  1
Card = 25  1
Card = 26  2
Card = 27  2
Card = 28  2
Card = 29  2
Card = 30  2
Card = 31  2
Card = 32  2
Card = 33  2
Card = 34  2
Card = 35  2
Card = 36  2
Card = 37  2
Card = 38  2
cont…
```
Card = 39  3
Card = 40  3
Card = 41  3
Card = 42  3
Card = 43  3
Card = 44  3
Card = 45  3
Card = 46  3
Card = 47  3
Card = 48  3
Card = 49  3
Card = 50  3
Card = 51  3

*/
Ex:

/* FILE: Cards3.c */
/* Simulate a deck of playing cards. */

- Produce the suit.

#include <stdio.h>

char suits[4][9] = { "Hearts",
                   "Diamonds",
                   "Clubs",
                   "Spades" };

int main()
{
    int i;
    for(i=0; i < 52; i++)
    {
        printf("Card = %2d %s\n", i, suits[i/13]);
    }
    return 0;
}

/* OUTPUT: Cards3.c */

Card = 0 Hearts
Card = 1 Hearts
Card = 2 Hearts
Card = 3 Hearts
Card = 4 Hearts
Card = 5 Hearts
Card = 6 Hearts
Card = 7 Hearts
Card = 8 Hearts
Card = 9 Hearts
Card = 10 Hearts
Card = 11 Hearts
Card = 12 Hearts
Card = 13 Diamonds
Card = 14 Diamonds
Card = 15 Diamonds
Card = 16 Diamonds
Card = 17 Diamonds
Card = 18 Diamonds
Card = 19 Diamonds
Card = 20 Diamonds
Card = 21 Diamonds
Card = 22 Diamonds
Card = 23 Diamonds
Card = 24 Diamonds
Card = 25 Diamonds
Card = 26 Clubs
Card = 27 Clubs
Card = 28 Clubs
Card = 29 Clubs
Card = 30 Clubs
Card = 31 Clubs
Card = 32 Clubs
Card = 33 Clubs
Card = 34 Clubs
Card = 35 Clubs
Card = 36 Clubs
Card = 37 Clubs
Card = 38 Clubs

cont…
A deck of cards

Card = 39 Spades
Card = 40 Spades
Card = 41 Spades
Card = 42 Spades
Card = 43 Spades
Card = 44 Spades
Card = 45 Spades
Card = 46 Spades
Card = 47 Spades
Card = 48 Spades
Card = 49 Spades
Card = 50 Spades
Card = 51 Spades

*/
Ex:

```c
/*   FILE: Cards4.c   */

/* Simulate a deck of playing cards.
   - Produce the suit & face values
     (kind of).
   */

#include <stdio.h>

char suits[4][9] = { "Hearts",
                    "Diamonds",
                    "Clubs",
                    "Spades" };

int main()
{
    int i;

    for(i=0; i < 52; i++){
        printf("Card = %2d %2d %s\n", i, i%13, suits[i/13]);
    }

    return 0;
}

/*   OUTPUT: Cards4.c   */

Card =  0  0 Hearts
Card =  1  1 Hearts
Card =  2  2 Hearts
Card =  3  3 Hearts
Card =  4  4 Hearts
Card =  5  5 Hearts
Card =  6  6 Hearts
Card =  7  7 Hearts
Card =  8  8 Hearts
Card =  9  9 Hearts
Card = 10 10 Hearts
Card = 11 11 Hearts
Card = 12 12 Hearts
Card = 13  0 Diamonds
Card = 14  1 Diamonds
Card = 15  2 Diamonds
Card = 16  3 Diamonds
Card = 17  4 Diamonds
Card = 18  5 Diamonds
Card = 19  6 Diamonds
Card = 20  7 Diamonds
Card = 21  8 Diamonds
Card = 22  9 Diamonds
Card = 23 10 Diamonds
Card = 24 11 Diamonds
Card = 25 12 Diamonds
Card = 26  0 Clubs

cont...
```
Card = 27 1 Clubs
Card = 28 2 Clubs
Card = 29 3 Clubs
Card = 30 4 Clubs
Card = 31 5 Clubs
Card = 32 6 Clubs
Card = 33 7 Clubs
Card = 34 8 Clubs
Card = 35 9 Clubs
Card = 36 10 Clubs
Card = 37 11 Clubs
Card = 38 12 Clubs
Card = 39 0 Spades
Card = 40 1 Spades
Card = 41 2 Spades
Card = 42 3 Spades
Card = 43 4 Spades
Card = 44 5 Spades
Card = 45 6 Spades
Card = 46 7 Spades
Card = 47 8 Spades
Card = 48 9 Spades
Card = 49 10 Spades
Card = 50 11 Spades
Card = 51 12 Spades
*/
Basic Multidimensional Arrays - examples

Ex:

/*     FILE: marray1.c     */
/*
Multidimensional arrays:
A 2-D array representing a table of
exam scores
*/
#include <stdio.h>
#define ROWS 10     /* Preprocessor directives to */
#define COLS 3      /* ...allow easy adjustment */
  /* ...table dimensions. */
int main( )
{  
  int row, col;       /* Variables for visiting/processing */
    /* ...every row & column in the table. */
  int scores[ROWS][COLS];
  for(row=0; row < ROWS; row++)  /* Zero out the table. */
    for(col=0; col < COLS; col++)
      scores[row][col] = 0;
    scores[0][0] = 90;  /* Place some sample values in the */
    scores[0][1] = 92;  /* ...for testing. */
    scores[0][2] = 93;
    scores[1][0] = 70;
    scores[1][1] = 89;
    scores[1][2] = 100;
    scores[2][0] = 85;
    scores[2][1] = 90;
    scores[2][2] = 95;
  for(row=0; row < ROWS; row++)    /* Nested loops to display */
    {                                /* ...the table. */
      for(col=0; col < COLS; col++)
        {  printf("  %d", scores[row][col]);
        }
      printf("\n");                  /* Add a newline after each */
    }                                /* ...row in the table. */
  return 0;
}
/* OUTPUT: marray1.c
  90  92  93
  70  89  100
  85  90  95
  0   0   0
  0   0   0
  0   0   0
  0   0   0
  0   0   0
  0   0   0
*/
Ex:

```c
/*     FILE: marray2.c     */
/**
Multidimensional arrays:
A 2-D array representing a table of exam scores
*/
#include <stdio.h>
#define ROWS 10       /* Preprocessor directives to */
#define COLS 3        /* ...allow easy adjustment */
/* ...table dimensions. */
int main( )
{
    int row, col;   /* Variables for visiting/processing */
    /* ...every row & column in the table. */
    int scores[ROWS][COLS];
    for(row=0; row < ROWS; row++)  /* Zero out the table. */
        for(col=0; col < COLS; col++)
            scores[row][col] = 0;
    scores[0][0] = 90;  /* Place some sample values in the */
    scores[0][1] = 92;  /* ...for testing.                 */
    scores[0][2] = 93;
    scores[1][0] = 70;
    scores[1][1] = 89;
    scores[1][2] = 100;
    scores[2][0] = 85;
    scores[2][1] = 90;
    scores[2][2] = 95;
    for(row=0; row < ROWS; row++)    /* Nested loops to display */
        {                                /* ...the table. */
            for(col=0; col < COLS; col++)
                { printf(" %5d", scores[row][col]);
                    }
            printf("\n");                  /* Add a newline after each */
        }                                /* ...row in the table. */
    return 0;
}

/* OUTPUT: marray2.c */
90  92  93
70  89  100
85  90  95
  0  0  0
  0  0  0
  0  0  0
  0  0  0
  0  0  0
  0  0  0
  0  0  0
*/
```
Ex:

```c
/*     FILE: marray3.c     */

/*
Multidimensional arrays:

A 2-D array representing a table of
exam scores
*/

#include <stdio.h>

#define ROWS 10       /* Preprocessor directives to */
#define COLS 3        /* ...allow easy adjustment */
                  /* ...table dimensions. */

int main( )
{
    int row, col;       /* Variables for visiting/processing */
                     /* ...every row & column in the table. */
    int j, sum;         /* Additional integer variables. */
    float avg;

    int scores[ROWS][COLS];

    for(row=0; row < ROWS; row++)  /* Zero out the table. */
        for(col=0; col < COLS; col++)
            scores[row][col] = 0;

    scores[0][0] = 90;  /* Place some sample values in the */
    scores[0][1] = 92;  /* ...for testing.                 */
    scores[0][2] = 93;

    scores[1][0] = 70;
    scores[1][1] = 89;
    scores[1][2] = 100;

    scores[2][0] = 85;
    scores[2][1] = 90;
    scores[2][2] = 95;

    for(row=0; row < ROWS; row++)    /* Nested loops to display */
        {                                /* ...the table.          */
            for(col=0; col < COLS; col++)
            {                            /* ...for each row.       */
                printf(" %5d", scores[row][col]);
            }

            for(sum=0, j=0; j<COLS; j++)  /* Compute and print average */
                {                        /* ...for each row.       */
                    sum += scores[row][j];
                }
            avg = (float)sum/COLS;
            printf(" %.2f", avg);
            printf("\n");                      /* Add a newline after each */
        }                                             /* ...row in the table. */

    return 0;
}

cont...
```
/* OUTPUT: marray3.c

90 92 93 91.67
70 89 100 86.33
85 90 95 90.00
0 0 0 0.00
0 0 0 0.00
0 0 0 0.00
0 0 0 0.00
0 0 0 0.00
0 0 0 0.00
*/
Ex:

```c
/*     FILE: marray4.c     */

/*
Multidimensional arrays:
A 2-D array representing a table of
exam scores
*/

#include <stdio.h>

#define ROWS 10       /* Preprocessor directives to */
#define COLS 3        /* ...allow easy adjustment */
                   /* ...table dimensions. */
int main( )
{
    int row, col;       /* Variables for visiting/processing */
                       /* ...every row & column in the table. */
    int j, sum;         /* Additional integer variables. */
    float avg;

    int scores[ROWS][COLS];

    for(row=0; row < ROWS; row++)  /* Zero out the table. */
    for(col=0; col < COLS; col++)
        scores[row][col] = 0;

    scores[0][0] = 90;  /* Place some sample values in the */
    scores[0][1] = 92;  /* ...for testing.                 */
    scores[0][2] = 93;

    scores[1][0] = 70;
    scores[1][1] = 89;
    scores[1][2] = 100;

    scores[2][0] = 85;
    scores[2][1] = 90;
    scores[2][2] = 95;

    for(row=0; row < ROWS; row++)    /* Nested loops to display */
    {                                /* ...the table. */
        for(col=0; col < COLS; col++)
        {
            printf(" %5d", scores[row][col]);
        }

        for(sum=0, j=0; j<COLS; j++)    /* Compute and print average */
        {                                /* ...for each row. */
            sum += scores[row][j];
        }

        avg = (float)sum/COLS;
        printf(" %10.2f", avg);        /* Adjust format for alignment. */
        printf("\n");                  /* Add a newline after each */
    }                                /* ...row in the table. */

    return 0;
}

cont...
/* OUTPUT: marray4.c

90  92  93  91.67
70  89 100  86.33
85  90  95  90.00
0   0   0   0.00
0   0   0   0.00
0   0   0   0.00
0   0   0   0.00
0   0   0   0.00
0   0   0   0.00
0   0   0   0.00
*/
Ex:
/*     FILE: marray5.c     */

/*
Multidimensional arrays:
A 2-D array representing a table of exam scores
*/
#include <stdio.h>
#define ROWS 10  /* Preprocessor directives to */
#define COLS 3   /* ...allow easy adjustment */
    /* ...table dimensions. */
int main( )
{
    int row, col;  /* Variables for visiting/processing */
        /* ...every row & column in the table. */
    int j, sum;    /* Additional integer variables. */
    float avg;

    int scores[ROWS][COLS];
    for(row=0; row < ROWS; row++)  /* Zero out the table. */
        for(col=0; col < COLS; col++)
            scores[row][col] = 0;
    scores[0][0] = 90;  /* Place some sample values in the */
    scores[0][1] = 92;  /* ...for testing. */
    scores[0][2] = 93;
    scores[1][0] = 70;
    scores[1][1] = 89;
    scores[1][2] = 100;
    scores[2][0] = 85;
    scores[2][1] = 90;
    scores[2][2] = 95;
    for(col=0; col < COLS; col++)  /* Add column headings. */
    {
        printf("%5s%d", "S", col+1);
    }
    printf(" %10s", "Average");
    printf("\n");
    for(row=0; row < ROWS; row++)  /* Nested loops to display */
    {                                /* ...the table. */
        for(col=0; col < COLS; col++)
        {
            printf(" %5d", scores[row][col]);
        }
        for(sum=0, j=0; j<COLS; j++) /* Compute and print average */
        {
            sum += scores[row][j];
        }
        avg = (float)sum/COLS;
        printf(" %10.2f", avg);
        printf("\n"); /* Add a newline after each */
    }
    return 0;
}
cont…
/* OUTPUT: marray5.c */

<table>
<thead>
<tr>
<th>S1</th>
<th>S2</th>
<th>S3</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>90</td>
<td>92</td>
<td>93</td>
<td>91.67</td>
</tr>
<tr>
<td>70</td>
<td>89</td>
<td>100</td>
<td>86.33</td>
</tr>
<tr>
<td>85</td>
<td>90</td>
<td>95</td>
<td>90.00</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.00</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.00</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.00</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.00</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.00</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.00</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.00</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.00</td>
</tr>
</tbody>
</table>
Ex:

```c
/*     FILE: marray6.c     */
/*
Multidimensional arrays:
A 2-D array representing a table of
exam scores
*/
#include <stdio.h>
#define ROWS 10       /* Preprocessor directives to */
#define COLS 3        /* ...allow easy adjustment */
/* ...table dimensions. */
int main( )
{
    int row, col;  /* Variables for visiting/processing */
    /* ...every row & column in the table. */
    int j, sum;    /* Additional integer variables. */
    float avg;

    int scores[ROWS][COLS];

    for(row=0; row < ROWS; row++)  /* Zero out the table. */
    for(col=0; col < COLS; col++)
        scores[row][col] = 0;
    scores[0][0] = 90;  /* Place some sample values in the */
    scores[0][1] = 92;  /* ...for testing. */
    scores[0][2] = 93;
    scores[1][0] = 70;
    scores[1][1] = 89;
    scores[1][2] = 100;
    scores[2][0] = 85;
    scores[2][1] = 90;
    scores[2][2] = 95;

    for(col=0; col < COLS; col++)  /* Dress-up headings. */
    {
        printf("%5s%d", "S", col+1);
    }
    printf(" %10s", "Average");
    printf("\n");

    for(col=0; col < COLS; col++)
    {
        printf(" %5s", "-----");
    }
    printf(" %10s", "-------");
    printf("\n");

    cont...
```
for(row=0; row < ROWS; row++) /* Nested loops to display */
{                                /* ... the table.      */
    for(col=0; col < COLS; col++)
    {                                /* ... display ... */
        printf(" %5d", scores[row][col]);
    }
    for(sum=0, j=0; j<COLS; j++) /* Compute and print average */
    {                               /* ... for each row. */
        sum += scores[row][j];
        avg = (float)sum/COLS;
        printf(" %10.2f", avg);
    }
    printf("\n");                  /* Add a newline after each */
    /* ... row in the table */
}

return 0;

/*
 OUTPUT: marray6.c

<table>
<thead>
<tr>
<th>S1</th>
<th>S2</th>
<th>S3</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>-----</td>
<td>----</td>
<td>----</td>
<td>---------</td>
</tr>
<tr>
<td>90</td>
<td>92</td>
<td>93</td>
<td>91.67</td>
</tr>
<tr>
<td>70</td>
<td>89</td>
<td>100</td>
<td>86.33</td>
</tr>
<tr>
<td>85</td>
<td>90</td>
<td>95</td>
<td>90.00</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.00</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.00</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.00</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.00</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.00</td>
</tr>
</tbody>
</table>
*/
Ex:

```c
#include <stdio.h>

int main( )
{
    int depth, row, col;
    int x[2][3][4];

    for(depth = 0; depth < 2; depth++)
        for(row = 0; row < 3; row++)
            for(col = 0; col < 4; col++)
                x[depth][row][col] = depth*100 + row*10 + col;

    printf("Array is: int x[2][3][4]\n");
    printf("------------------- ------\n");
    for(depth = 0; depth < 2; depth++)
    {
        printf(" Layer %d:\n", depth);
        printf(" ========\n");
        for(row = 0; row < 3; row++)
        {
            for(col = 0; col < 4; col++)
                printf(" x[%d][%d][%d] = %3.3d  ",
                        depth, row, col, x[depth][row][col]);
        }
    }
    return 0;
}
```

/*    OUTPUT: mArray7.c    */

Array is:  int x[2][3][4]
------------------------
Layer 0:
        =========
 x[0][0][0] = 000  x[0][0][1] = 001  x[0][0][2] = 002  x[0][0][3] = 003
 x[0][1][0] = 010  x[0][1][1] = 011  x[0][1][2] = 012  x[0][1][3] = 013
 x[0][2][0] = 020  x[0][2][1] = 021  x[0][2][2] = 022  x[0][2][3] = 023
Layer 1:
        =========
 x[1][0][0] = 100  x[1][0][1] = 101  x[1][0][2] = 102  x[1][0][3] = 103
 x[1][1][0] = 110  x[1][1][1] = 111  x[1][1][2] = 112  x[1][1][3] = 113
 x[1][2][0] = 120  x[1][2][1] = 121  x[1][2][2] = 122  x[1][2][3] = 123
*/
**MULTIDIMENSIONAL ARRAYS & POINTERS - EXAMPLES**

Ex:

```c
/* FILE: pointer3.c */

/* Arrays are sets of storage locations. */

Multidimensional arrays are sets of sets of storage locations.

An array name is the address of the first thing (or set) in the array. */

#include <stdio.h>

void displayArray(int ar[ ][3], int length);

int main( )
{
    int *ptr;       /* pointer to a integer */
    int (*ptr2)[3]; /* pointer to a set of 3 */
    int i, j;
    int x[5][3];    /* 5x3 array - or a set */
    ptr = x[0];     /* ... of 5, 3 integer */
    ptr2 = x;
    for (i=0; i<5; i++) /* Load array with */
    for (j=0; j<3; j++) /* ... recognizable values. */
        x[i][j] = i*10 + j;

    for (i=0; i<5; i++) /* Display address of each row. */
        printf("x + %d is at address %p\n", i, x + i);
    for (i=0; i<5; i++) /* Display address of 1st integer in */
        printf("x[%d] is at address %p\n", i, x[i]);

    printf("\n");
    for (i=0; i<3; i++) /* Compute address of next item in */
        printf("x + %d is at address %p\n", i, x[i] + i);
    for (i=0; i<3; i++) /* Compute address of next item in the */
        printf("x[%d]+%d is at address %p\n", 0, i, x[0] + i); /* ... particular */

    printf("\n");
    displayArray(x, 5); /* Display array contents. */
    return 0;
}

void displayArray(int ar[ ][3], int length)
{
    int i,j;
    for (i=0; i<length; i++)
    {
        for (j=0; j<3; j++)
            printf("ar[%d][%d] = %2.2d ", i, j, ar[i][j]);
        printf("\n");
    }
}

cont…
/* OUTPUT: pointer3.c

x + 0 is at address 0022FEF0
x + 1 is at address 0022FEFC
x + 2 is at address 0022FF08
x + 3 is at address 0022FF14
x + 4 is at address 0022FF20
x[0] is at address 0022FEF0
x[1] is at address 0022FEFC
x[2] is at address 0022FF08
x[3] is at address 0022FF14
x[4] is at address 0022FF20

x + 0 is at address 0022FEF0
x + 1 is at address 0022FEFC
x + 2 is at address 0022FF08
x[0]+0 is at address 0022FEF0
x[0]+1 is at address 0022FEF4
x[0]+2 is at address 0022FEF8

ar[0][0] = 00 ar[0][1] = 01 ar[0][2] = 02
ar[1][0] = 10 ar[1][1] = 11 ar[1][2] = 12
ar[3][0] = 30 ar[3][1] = 31 ar[3][2] = 32
ar[4][0] = 40 ar[4][1] = 41 ar[4][2] = 42

*/
Ex:
/*     FILE: pointer4.c     */
/* 3-D array name, what does it represent?     */
/* Arrays are sets of storage locations.         */
/* Multidimensional arrays are sets of sets      */
/* of storage locations.                        */
/* An array name is the address of the first     */
/* thing (or set) in the array.                  */
*
#include <stdio.h>

int main( )
{
    int depth, row, col;
    int x[2][3][4];
    for(depth = 0; depth < 2; depth++) /* Load array with     */
        for(row = 0; row < 3; row++) /* ...recognizable values */
            for(col = 0; col < 4; col++)
                x[depth][row][col] = depth*100 + row*10 + col;

    /* Display "sets" within the 3-D array with their addresses. */
    printf("n\nArray is: int x[2][3][4]\n";
    printf("n --- --- --- ---\n");
    for(depth = 0; depth < 2; depth++)
        for(row = 0; row < 3; row++)
            for(col = 0; col < 4; col++)
                printf(" x[\%d][\%d][\%d] = %3.3d
", depth, row, col, x[depth][row][col]);

    return 0;
}
/* OUTPUT: pointer4.c 

Array is: int x[2][3][4] 
--------------------------- 

x + 0 = 0022FED0 
=-=-=-=-=-=-=-=-

x[0] + 0 = 0022FED0 
---------------------

x[0][0]+0=0022FED0 x[0][0]+1=0022FED4 x[0][0]+2=0022FED8 x[0][0]+3=0022FEDC x[0][0][0] = 000 x[0][0][1] = 001 x[0][0][2] = 002 x[0][0][3] = 003 

x[0] + 1 = 0022FEE0 
---------------------

x[0][1]+0=0022FEE0 x[0][1]+1=0022FEE4 x[0][1]+2=0022FEE8 x[0][1]+3=0022FEEC x[0][1][0] = 010 x[0][1][1] = 011 x[0][1][2] = 012 x[0][1][3] = 013 

x[0] + 2 = 0022FEF0 
---------------------

x[0][2]+0=0022FEF0 x[0][2]+1=0022FEF4 x[0][2]+2=0022FEF8 x[0][2]+3=0022FEFC x[0][2][0] = 020 x[0][2][1] = 021 x[0][2][2] = 022 x[0][2][3] = 023 

x + 1 = 0022FF00 
=-=-=-=-=-=-=-=-

x[1] + 0 = 0022FF00 
---------------------

x[1][0]+0=0022FF00 x[1][0]+1=0022FF04 x[1][0]+2=0022FF08 x[1][0]+3=0022FF0C x[1][0][0] = 100 x[1][0][1] = 101 x[1][0][2] = 102 x[1][0][3] = 103 

x[1] + 1 = 0022FF10 
---------------------

x[1][1]+0=0022FF10 x[1][1]+1=0022FF14 x[1][1]+2=0022FF18 x[1][1]+3=0022FF1C x[1][1][0] = 110 x[1][1][1] = 111 x[1][1][2] = 112 x[1][1][3] = 113 

x[1] + 2 = 0022FF20 
---------------------

x[1][2]+0=0022FF20 x[1][2]+1=0022FF24 x[1][2]+2=0022FF28 x[1][2]+3=0022FF2C x[1][2][0] = 120 x[1][2][1] = 121 x[1][2][2] = 122 x[1][2][3] = 123 

*/

LINKED LISTS

• A disadvantage of an array is that it cannot expand & contract once it is allocated.
• To allow individual items to be added and deleted from a set, each element must be individually “linked” into the set so that it’s link can be easily removed or added.
• This type of data structure is called a linked list.
• Structures are tailor made for linked lists. A structure can be defined to contain the data and a link to another structure of the same type. These structures are then called “self-referential” structures since the definition of the structure refers to itself in defining the “link pointer” it contains.
• The start of the list is tracked by a separate stand-alone pointer, often referred to as the “head”, for head-of-the-list.
• The end of the list is indicated by the last link in the list pointing to NULL and not another structure.
• New elements can be added to the end of the list or can even be inserted into the list.
• Unwanted elements can be removed and the pointers can be reassigned around the removed link to maintain a legitimate list with no excess storage.
• The links are generally dynamically allocated so that new links are created as needed and unwanted links can be freed along with their resources.
• Some of the disadvantages of linked lists are that there is overhead in the allocation and deallocation, and there is additional house-keeping in maintaining the list.
Ex:

```c
/*     FILE: linkedList1.c     */

/*
Some linked list basics
- Building the list
- Traversing the list
- Searching the list
- Removing a node from the list
*/
#include <stdio.h>

struct node {              // A node in the list
    int value;          // - Data in the node
    struct node * next; // - Link in the node
};

int main( )
{
    int i;
    struct node * head = NULL;  // Head of the list
    struct node * tmpPointer;   // Two utility
    struct node * tmpPointer2;  // ...pointers

    for (i=0; i<5; i++)         // Build a list
    {
        if(head == NULL){
            head = malloc(sizeof(struct node));
            head->next = NULL;
        }
        else{
            tmpPointer = malloc(sizeof(struct node));
            tmpPointer->next = head;
            head = tmpPointer;
        }
        head->value = i*10 + 5;
    }

    tmpPointer = head;           // Traverse and display the list
    while(tmpPointer != NULL){
        printf("Value = %d \n", tmpPointer->value);
        tmpPointer = tmpPointer->next;
    }

    tmpPointer = head;           // Find the value 25 in the list
    while(tmpPointer != NULL && tmpPointer->value != 25){
        tmpPointer = tmpPointer->next;
    }

    if(tmpPointer != NULL){      // Remove a node from the list
        if(tmpPointer == head)
            head = tmpPointer->next;
        else{
            tmpPointer2 = head;      // Find previous node
            while(tmpPointer2->next != tmpPointer){
                tmpPointer2 = tmpPointer2->next;
            }
            // Adjust pointers around node
            tmpPointer2->next = tmpPointer->next;
        }
        free(tmpPointer);          // Remove the node
    }

    cont...
```

```
```
 Linked Lists

tmpPointer = head;  // Traverse and display the list
while(tmpPointer != NULL){
  printf("Value = %d \n", tmpPointer->value);
  tmpPointer = tmpPointer->next;
}

head = NULL;  // The list is lost

return 0;
}

/*  OUTPUT: linkedList1.c
    Value = 45
    Value = 35
    Value = 25
    Value = 15
    Value = 5
    Value = 45
    Value = 35
    Value = 15
    Value = 5
*/
Ex:

```c
/*     FILE: linkedList2_1.c     */
/*
Some linked list basics
- Building the list
- Adding to the front
- Adding to the end
- Functions for these jobs
*/
#include <stdio.h>
#include <stdlib.h>

struct pair{  /* (x,y) pair defined w/ a link*/
    int x;
    int y;
    struct pair * next;
};

void prependNode(struct pair **h, int x, int y);
void printList(struct pair *h);
void appendNode(struct pair **h, int x, int y);
void printNode(struct pair p);

int main(int argc, char** argv)
{
    /* pointer for head of pairs list */
    struct pair* head = NULL;

    /* Manually add first node/link
    ... to the list              */
    head = malloc(sizeof(struct pair));
    head->x = 5;
    head->y = 7;
    head->next = NULL;
    printList(head);

    /* Manually add another node/link
    ... to the list

    Note: This node/link is further
    ... along the list than
    ... the first node. (Hence
    ... the ->next in each line.)*/
    head->next = malloc(sizeof(struct pair));
    head->next->x = -6;
    head->next->y = 21;
    head->next->next = NULL;
    printList(head);

    /* Call function to append a
    ... node/link to the list   */
    appendNode(&head, 77, -135);
    printList(head);

    /* Call function to prepend a node/link
    ... to the front of the list */
    prependNode(&head, -1, -1);
    printList(head);
    return 0;
}
```

cont…
void printNode(struct pair p) {
    printf("(%d,%d)\n", p.x, p.y);
}

void printList(struct pair *h)
{
    for(;h != NULL; h = h->next)
        printNode(*h);
    printf("========\n");
}

void prependNode(struct pair **h, int x, int y)
{
    struct pair *newNode;
    newNode = malloc(sizeof(struct pair));
    newNode->x = x;  // Load the node
    newNode->y = y;
    newNode->next = *h;  // Link to front of list
    *h = newNode;  // Link head to prepended
                   // ... node
}

void appendNode(struct pair **h, int x, int y)
{
    struct pair *end;
    if(*h == NULL)
        end = *h = malloc(sizeof(struct pair));
    else{
            // Find the end of the list
        for(end=*h;end->next != NULL; end = end->next)
            ;
            // Make the node
        end->next = malloc(sizeof(struct pair));
        end = end->next;  // Move to the end node
    }
    end->x = x;  // Load the node
    end->y = y;
    end->next = NULL;
}

cont...
/*    OUTPUT: linkedList2_1.c

(5,7)
=======
(5,7)
(-6,21)
=======
(5,7)
(-6,21)
(77,-135)
=======
(-1,-1)
(5,7)
(-6,21)
(77,-135)
=======

*/
Ex:

/*     FILE: linkedList2_2.c     */

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

struct pair{  /* (x,y) pair defined w/ a link*/
    int x;
    int y;
    struct pair * next;
};

void prependNode(struct pair **h, int x, int y);
void printList(struct pair *h);
void appendNode(struct pair **h, int x, int y);
void printNode(struct pair p);
void freeList(struct pair *h);

int main(int argc, char** argv)
{
    /* pointer for head of pairs list */
    struct pair* head = NULL;

    /* Manually add first node/link ... to the list */
    head = malloc(sizeof(struct pair));
    head->x = 5;
    head->y = 7;
    head->next = NULL;
    printList(head);

    /* Manually add another node/link ... to the list
    Note: This node/link is further ...
    along the list than ...
    the first node. (Hence ...
    the ->next in each line.) */
    head->next = malloc(sizeof(struct pair));
    head->next->x = -6;
    head->next->y = 21;
    head->next->next = NULL;
    printList(head);

    /* Call function to append a ... node/link to the list */
    prependNode(&head, 77, -135);
    printList(head);

    /* Call function to prepend a node/link ... to the front of the list */
    appendNode(&head, -1, -1);
    printList(head);
    freeList(head);
    head = NULL;
    cont...
```c
printList(head);
return 0;
}

/*
 * printNode( ) displays the data from
 * ... one node/link
 */
void printNode(struct pair p)
{
    printf("(%d,%d)\n", p.x,p.y);
}

/*
 * printList( ) traverses the entire list and
 * ... displays each node/link using printNode( )
 */
void printList(struct pair *h)
{
    for(;h != NULL; h = h->next)
        printNode(*h);
    printf("========\n");
}

/*
 * freeList( ) traverses the entire list and
 * ... frees each node/link
 */
void freeList(struct pair *h)
{
    struct pair * del;
    while(h != NULL){ // Walk thru list to the end
        del = h;        // Mark node to delete
        h = h->next;    // Advance head pointer
        free(del);      // Delete the node
        printList(h);   // Display the current list
        // ... as deleting
    }
}

/*
 * prependNode( ) needs to alter the head pointer
 * ... so the parameter is a pointer-to-a-pointer
 */
void prependNode(struct pair **h, int x, int y)
{
    struct pair *newNode; // Make the node
    newNode = malloc(sizeof(struct pair));
    newNode->x = x;        // Load the node
    newNode->y = y;
    newNode->next = *h;
    *h = newNode;           // Link head to prepended
                           // ... node
}
```

/* appendNode( ) finds end of list and appends a ... new node/link to the list */
void appendNode(struct pair **h, int x, int y)
{
struct pair *end;

if(*h == NULL)
   end = *h = malloc(sizeof(struct pair));
else{
   // Find the end of the list
   for(end=*h;end->next != NULL; end = end->next)
      ;
   // Make the node
   end->next = malloc(sizeof(struct pair));
   end = end->next;  // Move to the end node
}
end->x = x;    // Load the node
end->y = y;
end->next = NULL;
}

/* OUTPUT: linkedList2_2.c

(5,7)
=======
(5,7)
(-6,21)
=======
(77,-135)
(5,7)
(-6,21)
=======
(77,-135)
(5,7)
(-6,21)
(-1,-1)
=======
(5,7)
(-6,21)
(-1,-1)
=======
(-6,21)
(-1,-1)
=======
(-1,-1)
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
=======
======
Ex:

/*     FILE: linkedList2_3.c     */
/*   Some linked list basics       */
/*     - CRITICAL!!: Freeing up the list */
/*     - CRITICAL!!: Make it easy/reliable for the programmer */

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

struct pair{  /* (x,y) pair defined w/ a link*/
    int x;
    int y;
    struct pair * next;
};

void prependNode(struct pair **h, int x, int y);
void printList(struct pair *h);
void appendNode(struct pair **h, int x, int y);
void printNode(struct pair p);
void freeList(struct pair **h);

int main(int argc, char** argv)
{
    /* pointer for head of pairs list */
    struct pair* head = NULL;
    /* Manually add first node/link      */
    /*   ... to the list               */
    head = malloc(sizeof(struct pair));
    head->x = 5;
    head->y = 7;
    head->next = NULL;
    printList(head);

    /* Manually add another node/link   */
    /*   ... to the list               */
    /*   Note: This node/link is further */
    /*   ... along the list than      */
    /*   ... the first node. (Hence    */
    /*   ... the ->next in each line.) */
    head->next = malloc(sizeof(struct pair));
    head->next->x = -6;
    head->next->y = 21;
    head->next->next = NULL;
    printList(head);

    /* Call function to append a      */
    /*   ... node/link to the list    */
    appendNode(&head, 77, -135);
    printList(head);

    /* Call function to prepend a node/link */
    /*   ... to the front of the list */
    prependNode(&head, -1, -1);
    printList(head);
    freeList(&head);
    cont…
printList(head);
return 0;
}

void printNode(struct pair p)
{
    printf("(%d,%d)\n", p.x,p.y);
}

void printList(struct pair *h)
{
    for(;h != NULL; h = h->next)
        printNode(*h);
    printf("========\n");
}

void freeList(struct pair **h)
{
    struct pair * del;
    while(*h != NULL){ // Walk thru list to the end
        del = *h;        // Mark node to delete
        *h = (*h)->next; // Advance head pointer
        free(del);
        printList(*h);   // Display the current list
    }
}

void prependNode(struct pair **h, int x, int y)
{
    struct pair *newNode;
    newNode = malloc(sizeof(struct pair));
    newNode->x = x;          // Load the node
    newNode->y = y;
    newNode->next = *h;
    *h = newNode;            // Link head to prepended
}

cont…
```c
/*
   appendNode( ) finds end of list and appends a
   ... new node/link to the list
*/
void appendNode(struct pair **h, int x, int y)
{
    struct pair *end;

    if(*h == NULL)
       end = *h = malloc(sizeof(struct pair));
    else{
        // Find the end of the list
        for(end=*h;end->next != NULL; end = end->next)
        ;
        // Make the node
        end->next = malloc(sizeof(struct pair));

        end = end->next;       // Move to the end node
    }
    end->x = x;              // Load the node
    end->y = y;
    end->next = NULL;
}

/* OUTPUT: linkedList2_3.c

(5,7)
=======
(5,7)
(-6,21)
=======
(5,7)
(-6,21)
(77,-135)
=======
(-1,-1)
(5,7)
(-6,21)
(77,-135)
=======
(5,7)
(-6,21)
(77,-135)
=======
(-6,21)
(77,-135)
=======
(77,-135)
=======
=======
=======
=======
=======
=======
=======

*/
```
Ex:

```c
/*     FILE: linkedList3.c     */
/*
   Linked list with a "dummy" node/link at the head of the list.
- Having a dummy node simplifies parameter passing since the
  head of the list will never change, there will always be
  an extra node at the start of the list. (Note: It does
  cost the space of the extra/wasted node. }
*/

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

struct pair{  /* (x,y) pair defined w/ a link*/
  int x;
  int y;
  struct pair * next;
};

void prependNode(struct pair *h, int x, int y);
void printList(struct pair *h);
void appendNode(struct pair *h, int x, int y);
void printNode(struct pair p);
void freeList(struct pair *h);

int main(int argc, char** argv)
{
  /* pointer for head of list with
    ... a dummy node/link           */
  struct pair* head = malloc(sizeof(struct pair));
  head->next = NULL;

  /* Manually add first node/link
    ... to the list              */
  head->next = malloc(sizeof(struct pair));
  head->next->x = 5;
  head->next->y = 7;
  head->next->next = NULL;
  printList(head);

  /* Manually add another node/link
    ... to the list
    Note: This node/link is further
    ... along the list than
    ... the first node. (Hence
    ... the ->next in each line.) */
  head->next->next = malloc(sizeof(struct pair));
  head->next->next->x = -6;
  head->next->next->y = 21;
  head->next->next->next = NULL;
  printList(head);

  /* Call function to append a
    ... node/link to the list    */
  appendNode(head, 77, -135);
  printList(head);
  cont...
```
/* Call function to prepend a node/link ... to the front of the list */
prependNode(head, -1, -1);

printList(head);
freeList(head);
printList(head);
return 0;
}

/* printNode( ) displays the data from ... one node/link */
void printNode(struct pair p)
{
    printf("(%d,%d)\n", p.x,p.y);
}

/*
printList( ) traverses the entire list and ... displays each node/link using printNode( ) */
void printList(struct pair *h)
{
    printf("===List===\n");
    h = h->next;
    for(;h != NULL; h = h->next)
        printNode(*h);
    printf("==========\n");
}

/*
freeList( ) traverses the entire list and ... frees each node/link */
void freeList(struct pair *h)
{
    struct pair * del;
    del = h->next; // Set the starting deletion point
    h->next = NULL; // Mark list as empty
    h = del; // Use local h to traverse list
    while(del != NULL){
        printf("Deleting: "); // Display deletions
        printNode(*h);
        h = h->next; // Move h to next node before deletion
        free(del); // Delete current first node
        del = h; // Move to next deletion point
    }
}

cont…
/ * prependNode( ) needs to alter the head pointer ... so the parameter is a pointer-to-a-pointer */
void prependNode(struct pair *h, int x, int y)
{
    struct pair *newNode;  // Make the node
    newNode = malloc(sizeof(struct pair));

    newNode->x = x;        // Load the node
    newNode->y = y;
    newNode->next = h->next; // Link to front of list

    h->next = newNode;      // Link head to prepended
                          // ... node
}

/* appendNode( ) finds end of list and appends a ... new node/link to the list */
void appendNode(struct pair *h, int x, int y)
{
    struct pair *end;    // Find the end of the list
    for(end=h;end->next != NULL; end = end->next)
    {
        end->next = malloc(sizeof(struct pair));

        end = end->next;    // Move to the end node
        end->x = x;         // Load the node
        end->y = y;
        end->next = NULL;
    }
}

/* OUTPUT: linkedList3.c

==List==
(5,7)
===========
==List==
(5,7)
(-6,21)
===========
==List==
(5,7)
(-6,21)
(77,-135)
===========
==List==
(-1,-1)
(5,7)
(-6,21)
(77,-135)
===========
Deleting: (-1,-1)
Deleting: (5,7)
Deleting: (-6,21)
Deleting: (77,-135)
==List==
===========
*/