

## sort1.c (Page 1 of 2)

```

1: /*-----*/
2: /* sort1.c */
3: /* Author: Bob Dondero */
4: /*-----*/
5:
6: #include <stdio.h>
7: #include <stdlib.h>
8: #include <assert.h>
9:
10: /*-----*/
11:
12: /* Sort pdNumbers[0..iCount-1] in ascending order. */
13:
14: static void insertionSort(double *pdNumbers, int iCount)
15: {
16:     int i1;
17:     int i2;
18:     double dTemp;
19:
20:     assert(pdNumbers != NULL);
21:
22:     for (i1 = 1; i1 < iCount; i1++)
23:         for (i2 = i1; i2 > 0; i2--)
24:             if (pdNumbers[i2] < pdNumbers[i2-1])
25:                 {
26:                     dTemp = pdNumbers[i2];
27:                     pdNumbers[i2] = pdNumbers[i2-1];
28:                     pdNumbers[i2-1] = dTemp;
29:                 }
30: }
31:
32: /*-----*/
33:
34: /* Read numbers from stdin, and write them in ascending order to
35:    stdout. Return 0 if successful, or EXIT_FAILURE if stdin contains
36:    a non-number. */
37:
38: int main(void)
39: {
40:     enum {ARRAY_LENGTH = 10};
41:
42:     double adNumbers[ARRAY_LENGTH];
43:     int iCount;
44:     double dNumber;
45:     int i;
46:     int iScanfRet;
47:
48:     /* Read the numbers into an array. */
49:     for (iCount = 0; iCount < ARRAY_LENGTH; iCount++)
50:     {
51:         iScanfRet = scanf("%lf", &dNumber);
52:         if (iScanfRet == 0)
53:             {
54:                 fprintf(stderr, "Non-number in stdin\n");
55:                 exit(EXIT_FAILURE);
56:             }
57:         if (iScanfRet == EOF)
58:             break;
59:         adNumbers[iCount] = dNumber;
60:     }
61:
62:     /* Sort the array. */
63:     insertionSort(adNumbers, iCount);

```

## sort1.c (Page 2 of 2)

```
64:
65:  /* Write the numbers from the array. */
66:  for (i = 0; i < iCount; i++)
67:      printf("%g\n", adNumbers[i]);
68:
69:  return 0;
70: }
```

## sort2.c (Page 1 of 2)

```

1: /*-----*/
2: /* sort2.c */
3: /* Author: Bob Dondero */
4: /*-----*/
5:
6: #include <stdio.h>
7: #include <stdlib.h>
8: #include <assert.h>
9:
10: /*-----*/
11:
12: /* Sort pdNumbers[0..iCount-1] in ascending order. */
13:
14: static void insertionSort(double *pdNumbers, int iCount)
15: {
16:     int i1;
17:     int i2;
18:     double dTemp;
19:
20:     assert(pdNumbers != NULL);
21:
22:     for (i1 = 1; i1 < iCount; i1++)
23:         for (i2 = i1; i2 > 0; i2--)
24:             if (pdNumbers[i2] < pdNumbers[i2-1])
25:                 {
26:                     dTemp = pdNumbers[i2];
27:                     pdNumbers[i2] = pdNumbers[i2-1];
28:                     pdNumbers[i2-1] = dTemp;
29:                 }
30: }
31:
32: /*-----*/
33:
34: /* The first number in stdin should be an integer which is a count of
35:    how many numbers follow in stdin. Read that integer. Then read the
36:    numbers from stdin, and write them in ascending order to stdout.
37:    Return 0 if successful, or EXIT_FAILURE if stdin contains a
38:    non-number. */
39:
40: int main(void)
41: {
42:     int iCount;
43:     int iArrayLength;
44:     double dNumber;
45:     int i;
46:     int iScanfRet;
47:
48:     /* Read the integer that specifies how many numbers follow. */
49:     iScanfRet = scanf("%d", &iArrayLength);
50:     if ((iScanfRet != 1) || (iArrayLength < 0))
51:     {
52:         fprintf(stderr, "Improper number count\n");
53:         exit(EXIT_FAILURE);
54:     }
55:
56:     /* Create an array with the specified number of elements. */
57:     double adNumbers[iArrayLength];
58:
59:     /* Read the numbers into the array. */
60:     for (iCount = 0; iCount < iArrayLength; iCount++)
61:     {
62:         iScanfRet = scanf("%lf", &dNumber);
63:         if (iScanfRet == 0)

```

## sort2.c (Page 2 of 2)

```
64:     {
65:         fprintf(stderr, "Non-number in stdin\n");
66:         exit(EXIT_FAILURE);
67:     }
68:     if (iScanfRet == EOF)
69:         break;
70:     adNumbers[iCount] = dNumber;
71: }
72:
73: /* Sort the array. */
74: insertionSort(adNumbers, iCount);
75:
76: /* Write the numbers from the array. */
77: for (i = 0; i < iCount; i++)
78:     printf("%g\n", adNumbers[i]);
79:
80: return 0;
81: }
82:
83: /*-----*/
84:
85: /* Sample build:
86:
87: $ gcc217 sort2.c
88: sort2.c: In function 'main':
89: sort2.c:57:4: warning:
90:     ISO C90 forbids variable length array 'adNumbers' [-Wvla]
91:     double adNumbers[iArrayLength];
92:     ^
93: sort2.c:57:4: warning:
94:     ISO C90 forbids mixed declarations and code
95:     [-Wdeclaration-after-statement]
96:
97: */
```

# Princeton University

## COS 217: Introduction to Programming Systems

### C Dynamic Memory Management Fundamentals

#### Dynamic Memory Management for Elementary Types

```
int *pi;
...
/* pi = (int*)malloc(4); */
pi = (int*)malloc(sizeof(int));
/* pi = (int*)malloc(sizeof(*pi)); */
...
*pi = 5;
...
free(pi);
...
```

#### Dynamic Memory Management for Arrays

```
int *pi;
...
/* pi = (int*)malloc(20); */
/* pi = (int*)malloc(5 * sizeof(int)); */
/* pi = (int*)malloc(5 * sizeof(*pi)); */
/* pi = (int*)calloc(5, 4); */
pi = (int*)calloc(5, sizeof(int));
/* pi = (int*)calloc(5, sizeof(*pi)); */
...
*(pi + 2) = 5;
pi[3] = 6;
...
free(pi);
...
```

#### Dynamic Memory Management for Structures

```
struct Location {int iLat; int iLon;};
...
struct Location *psLoc;
...
psLoc = (struct Location*)malloc(sizeof(struct Location));
/* psLoc = (struct Location*)malloc(sizeof(*psLoc)); */
...
(*psLoc).iLat = 41;
psLoc->iLon = 74;
...
free(psLoc);
...
```

#### Changing the Size of a Dynamically Allocated Array

```
int *pi;
...
pi = (int*)calloc(5, sizeof(int));
...
pi = (int*)realloc(pi, 3 * sizeof(int));
...
pi = (int*)realloc(pi, 10 * sizeof(int));
...
free(pi);
...
```

## sort3.c (Page 1 of 2)

```

1: /*-----*/
2: /* sort3.c */
3: /* Author: Bob Dondero */
4: /*-----*/
5:
6: #include <stdio.h>
7: #include <stdlib.h>
8: #include <assert.h>
9:
10: /*-----*/
11:
12: /* Sort pdNumbers[0..iCount-1] in ascending order. */
13:
14: static void insertionSort(double *pdNumbers, int iCount)
15: {
16:     int i1;
17:     int i2;
18:     double dTemp;
19:
20:     assert(pdNumbers != NULL);
21:
22:     for (i1 = 1; i1 < iCount; i1++)
23:         for (i2 = i1; i2 > 0; i2--)
24:             if (pdNumbers[i2] < pdNumbers[i2-1])
25:                 {
26:                     dTemp = pdNumbers[i2];
27:                     pdNumbers[i2] = pdNumbers[i2-1];
28:                     pdNumbers[i2-1] = dTemp;
29:                 }
30: }
31:
32: /*-----*/
33:
34: /* The first number in stdin should be an integer which is a count of
35:    how many numbers follow in stdin. Read that integer. Then read the
36:    numbers from stdin, and write them in ascending order to stdout.
37:    Return 0 if successful, or EXIT_FAILURE if stdin contains a
38:    non-number. */
39:
40: int main(void)
41: {
42:     double *pdNumbers;
43:     int iCount;
44:     int iArrayLength;
45:     double dNumber;
46:     int i;
47:     int iScanfRet;
48:
49:     /* Read the integer that specifies how many numbers follow. */
50:     iScanfRet = scanf("%d", &iArrayLength);
51:     if ((iScanfRet != 1) || (iArrayLength < 0))
52:     {
53:         fprintf(stderr, "Improper number count\n");
54:         exit(EXIT_FAILURE);
55:     }
56:
57:     /* Create an array with the specified number of elements. */
58:     pdNumbers = (double*)calloc((size_t)iArrayLength, sizeof(double));
59:     if (pdNumbers == NULL)
60:     {
61:         fprintf(stderr, "Cannot allocate memory\n");
62:         exit(EXIT_FAILURE);
63:     }

```

## sort3.c (Page 2 of 2)

```
64:
65:  /* Read the numbers into the array. */
66:  for (iCount = 0; iCount < iArrayLength; iCount++)
67:  {
68:      iScanfRet = scanf("%lf", &dNumber);
69:      if (iScanfRet == 0)
70:      {
71:          fprintf(stderr, "Non-number in stdin\n");
72:          exit(EXIT_FAILURE);
73:      }
74:      if (iScanfRet == EOF)
75:          break;
76:      pdNumbers[iCount] = dNumber;
77:  }
78:
79:  /* Sort the array. */
80:  insertionSort(pdNumbers, iCount);
81:
82:  /* Write the numbers from the array. */
83:  for (i = 0; i < iCount; i++)
84:      printf("%g\n", pdNumbers[i]);
85:
86:  /* Free the array. */
87:  free(pdNumbers);
88:
89:  return 0;
90: }
```

## sort4.c (Page 1 of 2)

```

1: /*-----*/
2: /* sort4.c */
3: /* Author: Bob Dondero */
4: /*-----*/
5:
6: #include <stdio.h>
7: #include <stdlib.h>
8: #include <assert.h>
9:
10: /*-----*/
11:
12: /* Grow the memory chunk pointed to by pdChunk so it can store
13:    uArrayLength elements of type double. Return the address of
14:    the new memory chunk. */
15:
16: static double *grow(double *pdChunk, size_t uArrayLength)
17: {
18:     double *pdNewChunk;
19:
20:     assert(pdChunk != NULL);
21:
22:     pdNewChunk =
23:         (double*)realloc(pdChunk, uArrayLength * sizeof(double));
24:     if (pdNewChunk == NULL)
25:     {
26:         fprintf(stderr, "Cannot allocate memory\n");
27:         exit(EXIT_FAILURE);
28:     }
29:
30:     return pdNewChunk;
31: }
32:
33: /*-----*/
34:
35: /* Sort pdNumbers[0..uCount-1] in ascending order. */
36:
37: static void insertionSort(double *pdNumbers, size_t uCount)
38: {
39:     size_t u1;
40:     size_t u2;
41:     double dTemp;
42:
43:     assert(pdNumbers != NULL);
44:
45:     for (u1 = 1; u1 < uCount; u1++)
46:         for (u2 = u1; u2 > 0; u2--)
47:             if (pdNumbers[u2] < pdNumbers[u2-1])
48:             {
49:                 dTemp = pdNumbers[u2];
50:                 pdNumbers[u2] = pdNumbers[u2-1];
51:                 pdNumbers[u2-1] = dTemp;
52:             }
53: }
54:
55: /*-----*/
56:
57: /* Read numbers from stdin, and write them in ascending order to
58:    stdout. Return 0 if successful, or EXIT_FAILURE if stdin contains
59:    a non-number. */
60:
61: int main(void)
62: {
63:     const size_t INITIAL_ARRAY_LENGTH = 2;

```



## sort4.c (Page 2 of 2)

```
64:     const size_t ARRAY_GROWTH_FACTOR = 2;
65:
66:     double *pdNumbers;
67:     size_t uCount;
68:     size_t uArrayLength;
69:     double dNumber;
70:     size_t u;
71:     int iScanfRet;
72:
73:     /* Create a small array. */
74:     uArrayLength = INITIAL_ARRAY_LENGTH;
75:     pdNumbers = (double*)calloc(uArrayLength, sizeof(double));
76:     if (pdNumbers == NULL)
77:     {
78:         fprintf(stderr, "Cannot allocate memory\n");
79:         exit(EXIT_FAILURE);
80:     }
81:
82:     /* Read the numbers into the array, expanding the size of the
83:     array as necessary. */
84:     for (uCount = 0; ; uCount++)
85:     {
86:         iScanfRet = scanf("%lf", &dNumber);
87:         if (iScanfRet == 0)
88:         {
89:             fprintf(stderr, "Non-number in stdin\n");
90:             exit(EXIT_FAILURE);
91:         }
92:         if (iScanfRet == EOF)
93:             break;
94:         if (uCount == uArrayLength)
95:         {
96:             uArrayLength *= ARRAY_GROWTH_FACTOR;
97:             pdNumbers = grow(pdNumbers, uArrayLength);
98:         }
99:         pdNumbers[uCount] = dNumber;
100:    }
101:
102:    /* Sort the array. */
103:    insertionSort(pdNumbers, uCount);
104:
105:    /* Write the numbers from the array. */
106:    for (u = 0; u < uCount; u++)
107:        printf("%g\n", pdNumbers[u]);
108:
109:    /* Free the array. */
110:    free(pdNumbers);
111:
112:    return 0;
113: }
```