CHAPTER 7

ARRAYS

reverse.c

(Not In Text)

```
/*
 * Read values and print them in reverse order.  This version doesn't
 * check for errors (such as the array filling up) and is not written
 * concisely.
 */
#include <stdio.h>

#define MAXVALS 100       /* max number of values we can reverse */

int main()
{
  int table[MAXVALS];     /* array to hold input values */
  int n;                  /* number of values in "table" */
  int i;                  /* index used in writing values */
  int next;               /* next input value */
  int r;                  /* return code from trying to read values */

  n = 0;
  while (scanf("%i", &next) == 1)      /* read values into table */
  {
    table[n] = next;
    n++;
  }
  for (i = n - 1; i >= 0; i--)         /* print table backwards */
    printf("%i\n", table[i]);
  return 0;
}
```

Sample I/O – revint

```
10 20 18 24 68 1 19
_ _ _ _ _
19
1
68
24
18
20
10
```

revint.c

(Figure 7.1 – Page 150)

```
/*
 * Read values and print them in reverse order.
 */
#include <stdio.h>

#define MAXVALS 100       /* max number of values we can reverse */

int main()
{
  int table[MAXVALS];     /* array to hold input values */
  int n;                  /* number of values in "table" */
  int i;                  /* index used in writing values */
  int next;               /* next input value */
  int r;                  /* return code from trying to read values */

  for (n = 0; (r = scanf("%i", &next)) != EOF; n++)
  {
    if (r != 1)           /* bad return from scanf */
    {
      printf("Input error after reading %i values.\n", n);
      break;
    }
    if (n == MAXVALS)     /* no room to store this value */
    {
      printf("No more room after reading %i values.\n", n);
      break;
    }
    table[n] = next;
  }

  for (i = n - 1; i >= 0; i--)
    printf("%i\n", table[i]);
  return 0;
}
```

revint2.c

(Figure 7.2 – Page 153)

```
/*
 * Read values and print them in reverse order.  This time uses:
 *   table_fill - reads values into an array.
 *   table_print_rev - prints the array in reverse order.
 */
#include <stdio.h>

#define MAXVALS 100         /* max number of values we can reverse */

int main()
{
  int  table_fill(int a[], int max);
  void table_print_rev(int a[], int max);
  int  table[MAXVALS];      /* array to hold input values */
  int  n;                   /* number of values in "table" */

  n = table_fill(table, MAXVALS);
  table_print_rev(table, n);
  return 0;
}
```

revint2.c

(Figure 7.2 – Page 153)

```
int table_fill(int a[], int max)
{
  int next;                    /* next input value */
  int r;                       /* return from trying to read values */
  int cnt;                     /* count of values read */

  for (cnt = 0; (r = scanf("%i", &next)) != EOF; cnt++)
  {
    if (r != 1)                /* bad return from scanf */
    {
      printf("Error in the input after reading %i values.\n", cnt);
      break;
    }
    if (cnt == max)            /* no room to store this value */
    {
      printf("No more room in array after reading %i values.\n", cnt);
      break;
    }
    a[cnt] = next;             /* save element in array */
  }
  return cnt;
}

void table_print_rev(int a[], int n)
{
  int i;                       /* index used in writing values */

  for (i = n - 1; i >= 0; i--)
    printf("%i\n", a[i]);
}
```

Sample I/O – usetable

```
10 45 67 16 0
67
Found 67 as element 2.
10
Found 10 as element 0.
76
Didn't find 76.
```

usetable.c

(Figure 7.3 – Page 156)

```
/*
 * Search table for the target value, returning its position.
 */
#include <stdio.h>

#define MAXVALS   10                    /* max # of entries in table */
#define SENTINEL   0                    /* terminating input value */

int main()
{
  int table_search(int a[], int n, int target);
  int table_fill_sentinel(int a[], int n, int termval);
  int table[MAXVALS];                   /* table of values */
  int target;                           /* value to find */
  int found;                            /* target position */
  int n;                                /* number of items */
  int r;                                /* scanf return value */

  n = table_fill_sentinel(table, MAXVALS, SENTINEL);
  while ((r = scanf("%i", &target)) == 1)
    if ((found = table_search(table, n, target)) != -1)
      printf("Found %i as element %i.\n", target, found);
    else
      printf("Didn't find %i.\n", target);
  if (r != EOF)
    printf("Illegal value to search for.\n");
  return 0;
}
```

tabsent.c

(Figure 7.4 – Page 157)

```
/*
 * Read values into an array until a sentinel value is read.
 */
#include <stdio.h>

int table_fill_sentinel(int a[], int max, int sentinel)
{
  int next;                       /* next input value */
  int r;                          /* return from trying to read values */
  int cnt;                        /* count of values read */

  cnt = 0;
  while ((r = scanf("%i", &next)) == 1 && next != sentinel)
  {
    if (cnt == max)               /* no room to store this value */
    {
      printf("Array full after reading %i values.\n", cnt);
      break;
    }
    a[cnt++] = next;              /* save element in array */
  }
  if (r == EOF)                   /* never saw sentinel */
    printf("End of file before encountering %i\n", sentinel);
  else if (r != 1)                /* bad return from scanf */
    printf("Error in the input after reading %i values.\n", cnt);
  return cnt;
}
```

tabsrch.c

(Figure 7.5 – Page 158)

```
/*
 * A function to search for a target in an array of integers.
 */
int table_search(int a[], int n, int target)
{
  int i;

  for (i = 0; i < n && a[i] != target; i++)
    ;                                   /* search for matching value */
  return (i != n) ? i : -1;
}
```

isort.c

(Figure 7.6 – Page 159)

```
/*
 * Read values and sort them using "insertion sort."  It uses:
 *   table_insert - place value in correct place in array.
 *   table_print - print array containing "num" items in sorted order.
 */
#include <stdio.h>

#define  MAXVALS  100                  /* max # of values to sort */

int main()
{
  void table_insert(int [], int, int);
  void table_print(int [], int);
  int  table[MAXVALS];                 /* table of values */
  int  n;                              /* number of values in table */
  int  r;                              /* value returned by scanf */
  int  v;                              /* current value */

  for (n = 0; (r = scanf("%i", &v)) != EOF; table_insert(table, n++, v))
  {
    if (r != 1)
    {
      printf("Input error after reading %i values\n", n);
      break;
    }
    if (n == MAXVALS)
    {
      printf("Table full after reading %i values\n", n);
      break;
    }
  }
  table_print(table, n);
  return 0;
}
```

isort.c

(Figure 7.6 – Page 159)

```
void table_insert(int a[], int num, int val)
{
  int pos;

  for (pos = num; pos > 0 && val < a[pos-1]; pos--)
    a[pos] = a[pos-1];
  a[pos] = val;
}

void table_print(int a[], int num)
{
  int i;

  for (i = 0; i < num; i++)
    printf("%i\n", a[i]);
}
```

HOW INSERTION SORT WORKS

table

[0]	[1]	[2]	[3]	[4]	[5]	
2	7	9	18	21		···

val	pos
8	5

(a) The value is smaller than table[pos-1], copy, decrement pos.

table

[0]	[1]	[2]	[3]	[4]	[5]	
2	7	9	18	21	21	···

val	pos
8	4

(b) The value is smaller than table[pos-1], copy, decrement pos.

table

[0]	[1]	[2]	[3]	[4]	[5]	
2	7	9	18	18	21	···

val	pos
8	3

(c) The value is smaller than table[pos-1], copy, decrement pos.

table

[0]	[1]	[2]	[3]	[4]	[5]	
2	7	9	9	18	21	···

val	pos
8	2

(d) Compare value with table[pos-1], value is bigger, install at table[pos].

table

[0]	[1]	[2]	[3]	[4]	[5]	
2	7	8	9	18	21	···

(e) The final array.

Sample Input – histo

```
89  87  56  89  67  78  79  80  85  97 100 100  23  45  12  14  87
88  84  84  84  84  77  77  72  73  68  69  69 100 100  11  75  71
71  72  79  84  71  72  70  74  75  73  71  79  72  55  55  68  65
67  72  71  75  73  71  74  71  71  78  79  80 100  91  92  66  61
61  57  57  65  71  74  78  78  78  78  78
```

Sample Output – histo

```
  0-  4 |
  5-  9 |
 10- 14 |***                        (3)
 15- 19 |
 20- 24 |*                          (1)
 25- 29 |
 30- 34 |
 35- 39 |
 40- 44 |
 45- 49 |*                          (1)
 50- 54 |
 55- 59 |*****                      (5)
 60- 64 |**                         (2)
 65- 69 |*********                  (9)
 70- 74 |*********************      (21)
 75- 79 |****************           (16)
 80- 84 |*******                    (7)
 85- 89 |******                     (6)
 90- 94 |**                         (2)
 95- 99 |*                          (1)
100-100 |*****                      (5)
```

histo.c

(Figure 7.10 – Page 163)

```
/*
 * Produce a histogram of its input values.
 */
#include <stdio.h>

#define NUMBCKTS   21            /* reasonable number of buckets  */
#define MINVAL      0            /* range of values is 0-100      */
#define MAXVAL    100            /*   (assumption is test scores) */

int main()
{
  void          fill_buckets(unsigned long buckets[],
                             int bucket_size, int numbuckets,
                             int minval, int maxval);
  void          print_histogram(unsigned long buckets[],
                                int bucket_size, int numbuckets,
                                int minval, int maxval);
  unsigned long buckets[NUMBCKTS];
  int           bucket_size;

  bucket_size = (MAXVAL - MINVAL) / (NUMBCKTS - 1);
  fill_buckets(buckets, bucket_size, NUMBCKTS, MINVAL, MAXVAL);
  print_histogram(buckets, bucket_size, NUMBCKTS, MINVAL, MAXVAL);
  return 0;
}
```

histo.c

(Figure 7.10 – Page 163)

```
/* Place each input value in an appropriate bucket */

void fill_buckets(unsigned long buckets[],
                  int bucket_size, int numbuckets,
                  int minval, int maxval)
{
  int inrange(int min, int max, int value);
  int next;                                /* next input value */
  int r;                                   /* scanf return value */
  int i;                                   /* index */

  for (i = 0; i < numbuckets; i++)         /* zero out buckets */
    buckets[i] = 0;
  while ((r = scanf("%i", &next)) != EOF)
    if (r != 1)
    {
      printf("Error while reading input values\n");
      break;
    }
    else if (!inrange(minval, maxval, next))
      printf("Out of range input value: %i\n", next);
    else
      buckets[(next - minval) / bucket_size]++;
}
```

histoout.c

(Figure 7.11 – Page 164)

```
/*
 * Print the histogram.
 */
#include <stdio.h>

#define MAXMARKS  25                 /* maximum marks per bucket */
#define MARKER    '*'                /* marker for values */

void print_histogram(unsigned long buckets[],
                     int bucketsize, int numbuckets,
                     int minval, int maxval)
{
  int           min(int x, int y);
  int           max(int x, int y);
  void          put_n_chars(char c, int n);
  unsigned long table_max(unsigned long a[], int n);
  int           b;                   /* bucket index */
  int           bmin;                /* smallest value in a bucket */
  int           bmax;                /* largest value in a bucket */
  double        scale;               /* scaling factor */
  int           marks;               /* # of marks to write on line */
  unsigned long most;                /* # of marks in largest bucket */

  most = table_max(buckets, numbuckets);
  scale = (most > MAXMARKS) ? (double) MAXMARKS / most : 1.0;
  for (b = 0, bmin = minval; b < numbuckets; b++, bmin = bmax + 1)
  {                                       /* print each row to scale */
    bmax = min(bmin + bucketsize - 1, maxval);
    marks = (buckets[b] > 0) ? max(buckets[b] * scale, 1) : 0;
    printf("%3i-%3i |", bmin, bmax);      /* write label */
    put_n_chars(MARKER, marks);           /* write markers */
    put_n_chars(' ', MAXMARKS - marks);   /* write spaces */
    if (buckets[b])                       /* write count */
      printf(" (%lu)", buckets[b]);
    putchar('\n');
  }
}
```

histoout.c

(Figure 7.11 – Page 164)

```
unsigned long table_max(unsigned long a[], int n)
{
  int           i;
  unsigned long largest;                  /* largest count */

  for (largest = 0, i = 0; i < n; i++)
    if (a[i] > largest)
      largest = a[i];
  return largest;
}
```

utils.c

(Figure 7.12 – Page 165)

```
/*
 * Some useful utility functions.
 *   min - returns smaller of two values.
 *   max - returns larger of two values.
 *   inrange - check whether one value is between two others.
 *   put_n_chars - write a character "n" times.
 */
#include <stdio.h>

int min(int x, int y)
{
  return (x < y) ? x : y;
}

int max(int x, int y)
{
  return (x > y) ? x : y;
}

int inrange(int min, int max, int v)
{
  return v >= min && v <= max;
}

void put_n_chars(char c, int n)
{
  while (n-- > 0)
    putchar(c);
}
```

REMINDERS

- Arrays are indexed starting with 0, not with 1 as in many other programming languages.

- Make sure any value used as array subscript is a legal array subscript.

- Changing an array parameter changes the corresponding array argument.

- **`sizeof`** returns the number of bytes in the array, not the number of elements.

CHAPTER SUMMARY

- C provides arrays whose size must be specified at compile time.

- Arrays are indexed from 0, and access outside defined bounds produces *undefined* behavior.

- When we pass an array to a function, only the address of its first element is copied, not the entire array. There's no need to specify the number of elements.

- Unlike the other variables we've seen, changes to array parameters affect the passed array.

- Some common operations on arrays are to read values into them, search for a value, sort the array, and print the values they contain.