CHAPTER 8

PROGRAM STRUCTURE

avg.c

(Figure 8.1 – Page 168)

```c
/*
 * Compute average of its input values with no error checking.
 */
#include <stdio.h>

int main()
{
  int    next, n;                    /* next input, number of inputs */
  long   sum = 0;                    /* running total */
  double avg;                        /* average of input values */

  for (n = 0; scanf("%i", &next) == 1; n++)
    sum += next;
  avg = (n == 0) ? 0.0 : (double) sum / n;
  printf("Average of %i values is %f.\n", n, avg);
  return 0;
}
```

Sample I/O – daycalc

```
Enter date (month day year): 6 24 1988
176
Enter date (month day year): 6 24 1991
175
Enter date (month day year): 2 19 1990
50
Enter date (month day year): 4 7 1992
98
Enter date (month day year): 1 1 1992
1
Enter date (month day year): 12 31 1991
365
Enter date (month day year): 12 31 1992
366
```

daycalc.c

(Figure 8.2 – Page 171)

```
/*
 * Given month, day, year, return day of year (no error checking).
 *   daycalc - calculate day or year, given month, day, and year.
 *   leapyear - returns 1 if current year a leap year, 0 otherwise.
 */
#include <stdio.h>

int main()
{
  int daycalc(int month, int day, int year);
  int month, day, year;                      /* current input date */

  while (printf("Enter date (month day year): "),
         scanf("%i %i %i", &month, &day, &year) == 3)
    printf("%i\n", daycalc(month,day,year));
  return 0;
}

int daycalc(int month, int day, int year)
{
  int leapyear(int year);
  int days[] = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
  int i;
  int total = day + (month > 2 && leapyear(year));

  for (i = 0; i < month - 1; total += days[i++])
    ;                          /* sum days in preceding months */
  return total;
}

int leapyear(int year)
{
  return year % 4 == 0 && year % 100 != 0 || year % 400 == 0;
}
```

avg2.c

(Figure 8.3 – Page 172)

```
/*
 * Compute average of its input values with no error checking.
 */
#include <stdio.h>

int main()
{
  int    next, n;                   /* next input, number of inputs */
  long   sum = 0;                   /* running total */

  for (n = 0; scanf("%i", &next) == 1; n++)
    sum += next;
  { /* compute and print input average */
    double avg = (n == 0) ? 0.0 : (double) sum / n;

    printf("Average of %i values is %f.\n", n, avg);
  }
  return 0;
}
```

Sample I/O – charcnt

```
this is a test
^D
```

```
\011    : 2
\012    : 1
\040 ( ): 2
\141 (a): 1
\145 (e): 1
\150 (h): 1
\151 (i): 2
\163 (s): 3
\164 (t): 3
```

charcnt.c

(Figure 8.4 – Page 173)

```
/*
 * A program to count different characters in its input.
 */
#include <stdio.h>
#include <ctype.h>
#include <limits.h>

unsigned long charcnts[UCHAR_MAX + 1];            /* 0..UCHAR_MAX */

int main()
{
  void print_counters(void);
  int  c;

  while(c = getchar(), c != EOF)
    charcnts[c]++;
  print_counters();
  return 0;
}

void print_counters(void)
{
  int i;

  for (i = 0; i < UCHAR_MAX; i++)
    if (charcnts[i])                  /* write count only when nonzero */
    {
      printf("\\%03o ", i);
      isprint(i) ? printf("(%c)", i) : printf("   ");
      printf(": %lu\n", charcnts[i]);
    }
}
```

tabsrch.c

(Figure 8.6 – Page 176)

```
/*
 * Faster version of array searching function.
 */
int table_search(int a[], register int n, register int t)
{
  register int i;

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

tstfnc1.c

(Figure 8.7 – Page 178)

```
/*
 * Keep a count of times function is called (buggy version).
 */
#include <stdio.h>

int main()
{
  void testfunc(void);

  testfunc(); testfunc(); testfunc();
  return 0;
}

void testfunc(void)
{
  int cnt = 0;

  printf("testfunc call #%i\n", ++cnt);
}
```

tstfnc2.c

(Figure 8.8 – Page 178)

```
/*
 * Keep a count of times function is called (working version).
 */
#include <stdio.h>

int main()
{
  void testfunc(void);

  testfunc(); testfunc(); testfunc();
  return 0;
}

void testfunc(void)
{
  static int cnt = 0;

  printf("testfunc call #%i\n", ++cnt);
}
```

pagenum.c

(Figure 8.9 – Page 179)

```
/*
 * Print lines with automatic page and line numbering.
 */
#include <stdio.h>

#define PAGELEN  60                          /* page length */

int main()
{
  void dumpchar(int c);
  int  c;                                   /* next input character */

  for (; (c = getchar()) != EOF; dumpchar(c))
    ;
  return 0;
}

void dumpchar(int c)
{
  static int pageno = 0;                    /* current page number */
  static int lastch = '\n';                 /* last char read */
  static int lineno = 0;                    /* current line number */

  if (lastch == '\n')                       /* beginning of line */
  {
    if (lineno % PAGELEN == 0)
      printf("\fPage: %i\n\n", ++pageno);  /* new page */
    printf("%6i ", ++lineno);
  }
  putchar(lastch = c);
}
```

charcnt1.c

(Figure 8.10 – Page 180)

```c
/*
 * A program to count the different characters in its input.  This
 * time it uses global variables to hold the counts.
 */
#include <stdio.h>
#include <limits.h>

unsigned long charcnts[UCHAR_MAX + 1];         /* 0..UCHAR_MAX */
int           chars = UCHAR_MAX + 1;

int main()
{
  void print_counters(void);
  int  c;

  while(c = getchar(), c != EOF)
    charcnts[c]++;
  print_counters();
  return 0;
}
```

prcnt1.c

(Figure 8.11 – Page 181)

```
/*
 * Function to print the character counts.
 */
#include <stdio.h>
#include <ctype.h>

void print_counters(void)
{
  extern unsigned long charcnts[];             /* table of counts */
  extern int           chars;                  /* entries in table */
  int                  i;

  for (i = 0; i < chars; i++)
    if (charcnts[i])                /* write count only when nonzero */
    {
      printf("\\%03o ", i);
      isprint(i) ? printf("(%c)", i) : printf("   ");
      printf(": %lu\n", charcnts[i]);
    }
}
```

charcnt2.c

(Figure 8.12 – Page 182)

```
/*
 * A new version of our character counting program.
 */
#include <stdio.h>
#include <limits.h>

unsigned long charcnts[UCHAR_MAX + 1];  /* 0..UCHAR_MAX-1 */
int           chars = UCHAR_MAX + 1;

void print_counters(void);

int main()
{
  int c;

  while(c = getchar(), c != EOF)
    charcnts[c]++;
  print_counters();
  return 0;
}
```

prcnt2.c

(Figure 8.13 – Page 182)

```
/*
 * A new version of the function to print the character counts.
 */
#include <stdio.h>
#include <ctype.h>

extern unsigned long charcnts[];  /* table of counts */
extern int           chars;       /* entries in table */

void print_counters(void)
{
  int i;

  for (i = 0; i < chars; i++)
    if (charcnts[i])     /* write count only when nonzero */
    {
      printf("\\%03o ", i);
      isprint(i) ? printf("(%c)", i) : printf("   ");
      printf(": %lu\n", charcnts[i]);
    }
}
```

charcnt3.c

(Figure 8.14 – Page 184)

```
/*
 * One final version of the character counting program.
 */
#include <stdio.h>
#include <ctype.h>
#include <limits.h>

int main()
{
  void   print_counters(const unsigned long charcnts[]);
  static unsigned long charcnts[UCHAR_MAX + 1];     /* 0..UCHAR_MAX */
  int    c;

  while(c = getchar(), c != EOF)
    charcnts[c]++;
  print_counters(charcnts);
  return 0;
}

void print_counters(const unsigned long charcnts[])
{
  int i;

  for (i = 0; i < UCHAR_MAX; i++)
    if (charcnts[i])              /* write count only when nonzero */
    {
      printf("\\%03o ", i);
      isprint(i) ? printf("(%c)", i) : printf("   ");
      printf(": %lu\n", charcnts[i]);
    }
}
```

yesorno.c

(Figure 8.15 – Page 186)

```
/*
 * Get a yes or no answer from the user.
 */
#include <stdio.h>
#include <ctype.h>

const int YES = 1, NO = 0;

int yesorno(void)
{
  int answer;                            /* holds input character */

  while ((answer = tolower(getchar())) != EOF)
  {
    int junk = answer;                   /* for skipping characters */

    while (junk != '\n' && junk != EOF)
      junk = getchar();                  /* skip until end of line */
    if (answer == 'y' || answer == 'n')
      return (answer == 'y') ? YES : NO;
    printf("Please answer with a YES or NO!\n");
  }
  return NO;
}
```


useyn1.c
(Figure 8.16 – Page 186)

```
/*
 * Main program using yes or no.
 */
#include <stdio.h>

int main()
{
  int              yesorno(void);
  extern const int YES, NO;

  printf("Enter a YES or NO answer: ");
  (yesorno() == YES) ? printf("YES!\n") : printf("NO!\n");
  return 0;
}
```

yesorno.h

(Figure 8.17 – Page 187)

```
/*
 * Prototypes and constants for "yesorno".
 */
extern const int YES;
extern const int NO;

int yesorno(void);
```

useyn2.c

(Figure 8.18 – Page 187)

```
/*
 * Main program using yes or no.
 */
#include <stdio.h>
#include "yesorno.h"                    /* for yesorno prototype */

int main()
{
  printf("Enter a YES or NO answer: ");
  (yesorno() == YES) ? printf("YES!\n") : printf("NO!\n");
  return 0;
}
```

queues.h

(Figure 8.19 – Page 189)

```
/*
 * Prototypes for our queue functions.
 */
void enqueue(int item);               /* add item to queue */
int  dequeue(void);                   /* take item away from queue */
int  emptyqueue(void);                /* is queue empty? */
```

queues.c

(Figure 8.20 – Page 189)

```
/*
 * Simple queue manager, no error checking.
 */
#include "queues.h"                   /* queue prototypes */

#define MAXQUEUE 100                  /* number of queue items */

static int queue[MAXQUEUE];           /* the queue itself */
static int f, r;                      /* indices to queue front and end */

static int next(int i);

void enqueue(int item)                /* add item to end of queue */
{
  queue[r] = item;
  r = next(r);
}

int dequeue(void)                     /* delete first item in queue */
{
  int temp = queue[f];

  f = next(f);
  return temp;
}

int emptyqueue(void)                  /* is queue empty? */
{
  return f == r;
}

/* PRIVATE: returns index to next item, handling wraparound */

static int next(int i)
{
  return (i + 1 < MAXQUEUE) ? i + 1 : 0;
}
```

wait.c

(Figure 8.21 – Page 192)

```
/*
 * Compute average waiting time of riders of a bus.
 */
#include <stdio.h>
#include "wait.h"
#include "handle.h"

unsigned long riders;                    /* total riders */
unsigned long waiters;                   /* total waiting time */

int main()
{
  int  code,                             /* transaction code */
       time,                             /* event time */
       count;                            /* room on bus, people at stop */

  while (scanf("%i %i %i", &code, &time, &count) == 3)
    switch(code)
    {
      case 0:  person(time, count);
               break;
      case 1:  bus(time, count);
               break;
      default: printf("Invalid code of %i\n", code);
               break;
    }
  if (riders == 0)
    printf("No riders ever showed up.\n");
  else
    printf("%lu riders waited %lu, average wait %.2f\n",
           riders, waiters, (double) waiters / riders);
  return 0;
}
```

Sample I/O – wait

```
0 2 2
2 people arrived at time 2
0 6 1
1 people arrived at time 6
0 7 4
4 people arrived at time 7
1 7 5
Bus arrived at time 7 (5 seats)
0 8 2
2 people arrived at time 8
0 8 9
9 people arrived at time 8
1 15 13
Bus arrived at time 15 (13 seats)
18 riders waited 104, average wait 5.78
```

wait.h

(Figure 8.22 – Page 192)

```
/*
 * Declarations for shared counters.
 */
extern unsigned long riders;            /* people on bus */
extern unsigned long waiters;           /* people waiting for bus */
```

handle.c

(Figure 8.23 – Page 193)

```
/*
 * Functions to update riders and waiting time.
 *   person - add a new set of riders to a bus stop.
 *   bus - take away a set of riders from a bus stop.
 */
#include <stdio.h>
#include "queues.h"
#include "wait.h"
#include "handle.h"

void person(int time, int newpeople)
{
  riders += newpeople;
  printf("%i people arrived at time %i\n", newpeople, time);
  while (newpeople--)
    enqueue(time);
}

void bus(int time, int capacity)
{
  printf("Bus arrived at time %i (%i seats)\n", time, capacity);
  while (capacity-- && !emptyqueue())
    waiters += time - dequeue();
}
```

handle.h

(Figure 8.24 – Page 193)

```
/*
 * Prototypes for functions to update appropriate counters.
 */
void person(int time, int persons);
void bus(int time, int capacity);
```

sets.h

(Figure 8.26 – Page 195)

```
/*
 * Definitions to use "sets" of integers.
 */
#define US_BITS    16           /* bits in unsigned short */
#define MAXELEMS  512           /* set items, must be multiple of 16 */

typedef unsigned short SET[MAXELEMS/US_BITS];

extern void set_init(SET s);
extern void set_add(SET s, int e);
extern void set_delete(SET s, int e);
extern int  set_member(const SET s, int e);
```

sets.c

(Figure 8.27 – Page 196)

```
/*
 * The functions to handle sets.
 */
#include "sets.h"

static int word(int elem)
{
  return elem / US_BITS;                /* word # containing element */
}

static int bit(int elem)
{
  return elem % US_BITS;                /* bit # containing element */
}

void set_init(SET s)                    /* initialize set to empty */
{
  int i;

  for (i = 0; i < MAXELEMS / US_BITS; s[i++] = 0)
    ;
}

void set_add(SET s, int elem)           /* add element to set */
{
  s[word(elem)] |= 1 << bit(elem);
}

void set_delete(SET s, int elem)        /* delete element from set */
{
  s[word(elem)] &= ~(1 << bit(elem));
}

int set_member(const SET s, int elem)   /* is element in set? */
{
  return ((unsigned) s[word(elem)] >> bit(elem)) & 01;
}
```

Sample I/O – usesets

```
8 99 99 245 0 99 501 99 309 17 410 17
_ _ _ _ _
Unique values
0
8
245
309
410
501
Duplicate values
17
99
```

usesets.c

(Figure 8.28 – Page 197)

```
/*
 * Identify duplicates in the input.
 */
#include <stdio.h>
#include "sets.h"                /* for SETs */

int main()
{
  void set_print(const SET s);
  void set_switch(SET news, SET olds, int elem);
  SET  unique, dup;              /* unique and duplicate elements */
  int  r, inp;                   /* scanf return, input value */

  set_init(unique);  set_init(dup);
  while ((r = scanf("%i", &inp)) != EOF)
  {
    if (r != 1)
    {
      printf("Value in error.\n");
      break;
    }
    if (inp < 0 || inp >= MAXELEMS)
    {
      printf("Value out of range.\n");
      continue;
    }
    if (set_member(unique, inp))
      set_switch(dup, unique, inp);
    else if (!set_member(dup, inp))
      set_add(unique, inp);
  }
  printf("Unique values\n");  set_print(unique);
  printf("Duplicate values\n");  set_print(dup);
  return 0;
}
```

usesets.c

(Figure 8.28 – Page 197)

```
void set_switch(SET new, SET old, int value)
{
  set_delete(old, value);  set_add(new, value);
}

void set_print(const SET set)
{
  int i;                              /* next potential element */

  for (i = 0; i < MAXELEMS; i++)
    if (set_member(set, i))
      printf("%i\n", i);
}
```

REMINDERS

- Don’t count on local variables automatically being initialized to zero.
- Make sure the array size you provide is large enough to hold the initial elements.
- Avoid global variables wherever possible.
- Define a global variable exactly once and use external declarations everywhere else.

CHAPTER SUMMARY

- C lets us declare variables in any block. These variables have visibility only within that block and exist only throughout the life of the block.

- C also lets us declare variables outside of any function. These variables are visible throughout the remainder of the source file. They can also be accessed from other files through the use of external declarations.

- C lets us initialize variables when we declare them. With local variables, we can provide any expression. With global variables, we're restricted to those expressions that are evaluable at compile time.

- C allows us to specify that variables are constants. When we do so, we must initialize them when we declare them.

CHAPTER SUMMARY

- We can place external declarations and constant definitions in header files that are included by the modules requiring them.

- We can have variables and functions local to a single source file by declaring them outside of any function and prefacing them with the storage class **`static`**.

- We implement abstract data types by defining the operations in a single source file and putting their external declarations in a header file. This way the functions using the data type aren't privy to the details of their implementation.