Fall 2006 Handout 7

How not to program

```
 1 #include <stdio.h>
 2 #include <stdlib.h>
 3
 4 main()
 5 {
 6     printf("There's a man who lives a life of danger\n");
 7     printf("To everyone he meets he stays a stranger\n");
 8     printf("With every move he makes, another chance he takes\n");
 9     printf("Odds are he won't live to see tomorrow.\n\n");
10
11     printf("Secret Agent Man, Secret Agent Man\n");
12     printf("They've given you a number, and taken 'way your name.\n");
13     printf("Secret Agent Man, Secret Agent Man\n");
14     printf("They've given you a number, and taken 'way your name.\n\n");
15
16     printf("Beware of pretty faces that you find\n");
17     printf("A pretty face can hide an evil mind\n");
18     printf("Be careful what you say, or you'll give yourself away\n");
19     printf("Odds are he won't live to see tomorrow.\n\n");
20
21     printf("Secret Agent Man, Secret Agent Man\n");
22     printf("They've given you a number, and taken 'way your name.\n");
23     printf("Secret Agent Man, Secret Agent Man\n");
24     printf("They've given you a number, and taken 'way your name.\n\n");
25
26     printf("Swinging on the Riviera one day\n");
27     printf("Lying in a Bombay alley the next day\n");
28     printf("Don't let the wrong words slip while kissing persuasive lips\n");
29     printf("Odds are he won't live to see tomorrow.\n\n");
30
31     printf("Secret Agent Man, Secret Agent Man\n");
32     printf("They've given you a number, and taken 'way your name.\n");
33     printf("Secret Agent Man, Secret Agent Man\n");
34     printf("They've given you a number, and taken 'way your name.\n\n");
35
36     exit(EXIT_SUCCESS);
37 }
```

A function: K&R pp. 24−27; King pp. 12−13, 155−176

The functions of a C program may be written in any order, but it's traditional to write the **`main`** function first. There must be exactly one **`main`** function. The program begins its execution at the first statement of the **`main`** function.

The function declaration in line 4 is exactly the same as the first line of the function definition in line 34, with the addition of a semicolon. Don't type this line twice: use your text editor to copy it for you. A function does not always need a **`return`** statement.

```
 1 #include <stdio.h>
 2 #include <stdlib.h>
 3
 4 void chorus(void);                  /* Line 4 is a function declaration. */
 5
 6 main()
 7 {
 8     printf("There's a man who lives a life of danger\n");
 9     printf("To everyone he meets he stays a stranger\n");
10     printf("With every move he makes, another chance he takes\n");
11     printf("Odds are he won't live to see tomorrow.\n\n");
12
13     chorus();
14
15     printf("Beware of pretty faces that you find\n");
16     printf("A pretty face can hide an evil mind\n");
17     printf("Be careful what you say, or you'll give yourself away\n");
18     printf("Odds are he won't live to see tomorrow.\n\n");
19
20     chorus();
21
22     printf("Swinging on the Riviera one day\n");
23     printf("Lying in a Bombay alley the next day\n");
24     printf("Don't let the wrong words slip while kissing persuasive lips\n");
25     printf("Odds are he won't live to see tomorrow.\n\n");
26
27     chorus();
28
29     exit(EXIT_SUCCESS);
30 }
31
32 /* Print the chorus. */
33
34 void chorus(void)                    /* Lines 34-40 are a function definition. */
35 {
36     printf("Secret Agent Man, Secret Agent Man\n");
37     printf("They've given you a number, and taken 'way your name.\n");
38     printf("Secret Agent Man, Secret Agent Man\n");
39     printf("They've given you a number, and taken 'way your name.\n\n");
40 }
```

A two-dimensional array of strings

Suppose you wanted to indent each line or double-space the lines: which version would be easier to change?

—On the Web at
`http://i5.nyu.edu/~mm64/x52.9232/src/agent2.c`

```
 1 #include <stdio.h>
 2 #include <stdlib.h>
 3
```

```
 4 void chorus(void);
 5
 6 main()
 7 {
 8     char *verse[][4] = {
 9         {
10             "There's a man who lives a life of danger",
11             "To everyone he meets he stays a stranger",
12             "With every move he makes, another chance he takes",
13             "Odds are he won't live to see tomorrow."
14         },
15
16         {
17             "Beware of pretty faces that you find",
18             "A pretty face can hide an evil mind",
19             "Be careful what you say, or you'll give yourself away",
20             "Odds are he won't live to see tomorrow."
21         },
22
23         {
24             "Swinging on the Riviera one day",
25             "Lying in a Bombay alley the next day",
26             "Don't let the wrong words slip while kissing persuasive lips",
27             "Odds are he won't live to see tomorrow."
28         }
29     };
30 #define N    (sizeof verse / sizeof verse[0])
31
32     int v;
33     int i;
34
35     for (v = 0; v < N; ++v) {
36         for (i = 0; i < 4; ++i) {
37             printf("%s\n", verse[v][i]);
38         }
39         chorus();
40     }
41
42     exit(EXIT_SUCCESS);
43 }
44
45 /* Print the chorus. */
46
47 void chorus(void)
48 {
49     int i;
50
51     printf("\n");
52
53     for (i = 0; i < 2; ++i) {
54         printf("Secret Agent Man, Secret Agent Man\n");
55         printf("They've given you a number, and taken 'way your name.\n");
56     }
57
```

```
58      printf("\n");
59  }
```

▼ **Homework 7.1: Write a function**

Write a program whose **main** function calls another function two or more times. A program that prints the lyrics to a song with verses and a chorus would be just fine. Use any of the following, or write your own.

''Ticket to Ride'', ''Yellow Submarine'', ''Help!'', ''Strawberry Fields Forever'', ''Ob-La-Di, Ob-La-Da'', all by the Beatles; ''Brown Eyed Girl'' by Van Morrison; ''Somebody to Love'' by the Jefferson Airplane; ''Tambourine Man'' by Bob Dylan; ''Puff, the Magic Dragon'' by the Weavers; ''I Told the Witch Doctor'' by the Three Chipmunks; ''Crocodile Rock'' by Elton John.

▲

▼ **Homework 7.2: examine the function declarations in the .h files (not to be handed in)**

Find the declarations for the functions **printf** and **scanf** in the file **stdio.h**, the functions **rand**, **srand** and **exit** in **stdlib.h**, the function **sqrt** in **math.h**, etc. For the **...** in the declarations of **printf** and **scanf**, see K&R pp. 155−156, King pp. 563−566.

On i5, the C **.h** files are in the directory **/usr/include** and its subdirectories; the C++ **.h** files are in the directory **/usr/local/lib/g++-include** and its subdirectories.

▲

A function with a variable: K&R pp. 24−27; King pp. 155−176

Build you own **sleep** or **delay** function (Handout 1, p. 21) if you version of C doesn't have it. You can declare variables after the **{** at the start of any function, but a variable declared there can be used only within that function. This means that you can have two variables with the same name if each is declared in a different function. See K&R p. 25, paragraph 3; King pp. 185−187.

```
 1  #include <stdio.h>
 2  #include <stdlib.h>
 3
 4  void sleeper(void);
 5
 6  main()
 7  {
 8      int i;
 9
10      printf("Prepare for countdown.\n");
11      sleeper();
12
13      for (i = 10; i >= 0; --i) {
14          printf("%d\n", i);
15          sleeper();
16      }
17
18      printf("Blast off!\n");
19      exit(EXIT_SUCCESS);
20  }
21
22  /* Delay for a brief time. */
23
24  void sleeper(void)
25  {
26      long i;
```

```
27
28      for (i = 0; i < 10000; ++i) {
29      }
30  }
```

```
10              portentous pause
9
8
7
6
5
4
3
2
1
0
Blast off!
```

More examples of a function with a variable

The following are function definitions. They are not complete C programs because no C program is complete without a function called **main**.

```
 1  /* Draw a horizontal line of 80 dashes. */
 2
 3  void line(void)
 4  {
 5      int i;
 6
 7      for (i = 0; i < 80; ++i) {
 8          putchar('-');                          /* #include <stdio.h> */
 9      }
10      putchar('\n');
11  }
```

```
 1  /* Pause until the user is ready to proceed or quit. */
 2
 3  void proceed(void)
 4  {
 5      char s[80];
 6
 7      printf("Press RETURN to proceed, or q RETURN to quit.\n");
 8      scanf("%s", s);                            /* #include <stdio.h> */
 9      if (strcmp(s, "q") == 0) {                 /* #include <string.h> */
10          exit(EXIT_SUCCESS);                    /* #include <stdlib.h> */
11      }
12  }
```

A function that can count how many times it's been called

The **long i** in the **sleeper** function above always ends its life with the value 10000. If **sleeper** is called again, however, the **long i** would be reborn with an unpredictable value. To make a variable retain its value between successive calls of its function, declare it to be **static**. See K&R p. 83, paragraph 4; King pp. 403−404.

```
 1 #include <stdio.h>
 2 #include <stdlib.h>
 3
 4 void counter(void);
 5
 6 main()
 7 {
 8     counter();
 9     counter();
10     counter();
11     exit(EXIT_SUCCESS);
12 }
13
14 void counter(void)
15 {
16     static int i = 0;        /* Initialized only once. */
17
18     printf("This is number %d.\n", ++i);
19 }
```

```
This is number 1.
This is number 2.
This is number 3.
```

Another function with a static variable

Suppose you are going to call the function **f** more than once, and there is a task it must perform only the first time that it is called.

```
 1 void f(void)
 2 {
 3     static int first = 1;
 4
 5     if (first) {
 6         printf("This is the first time I am called.\n");
 7         first = 0;
 8     }
 9
10     printf("I print this message every time I am called.\n");
11 }
```

How to `return` from a function: K&R pp. 26, 70; King pp. 169−171

The **`return`** statement makes you return from a function. In addition, reaching the **`}`** at the end of the function will also make you return. For this reason, the **`return`** statement is never written at the end of a function.

Write **`return`** only if you want to return *before* reaching the **`}`** at the end of the function. There can even be several **`return`** statements in a function, but it's less error prone to return only from the **`}`**.

```
 1 #include <stdio.h>
 2 #include <stdlib.h>
 3 #include <time.h>
 4
 5 void mislead(void);
 6
```

```
 7 main()
 8 {
 9     srand(time(NULL));
10     mislead();
11     exit(EXIT_FAILURE);
12 }
13
14 /* Output what looks like an error message and a bogus error number. */
15
16 void mislead(void)
17 {
18     int r = rand() % 3;          /* Let r be 0, 1, or 2. */
19
20     if (r == 0) {
21         printf("core dumped\n");
22         printf("error code %d\n", r);
23         return;
24     } else if (r == 1) {
25         printf("memory fault\n");
26         printf("error code %d\n", r);
27         return;
28     } else {
29         printf("segmentation violation\n");
30         printf("error code %d\n", r);
31         return;
32     }
33 }
```

```
memory fault
error code 1
```

```
34 /* Better version of the function definition, with code sinking and only one
35 point of return. */
36
37 void mislead(void)
38 {
39     int r = rand() % 3;
40
41     if (r == 0) {
42         printf("core dumped\n");
43     } else if (r == 1) {
44         printf("memory fault\n");
45     } else {
46         printf("segmentation violation\n");
47     }
48
49     printf("error code %d\n", r);
50 }
```

Homework 6.3.1 eliminates the three-way **if** by storing the three messages in an array of strings.

Return value

If the function yields a return value (unlike the ones shown above), you cannot return automatically from the **}**. You must always write the word **return** followed by an expression. Write the data type of the return value in place of the word **void** at the start of the function declaration and definition. See K&R p. 25; King pp. 164−165.

```
 1 #include <stdio.h>
 2 #include <stdlib.h>
 3 #include <time.h>
 4
 5 int rand100(void);
 6
 7 main()
 8 {
 9     int i;
10
11     srand(time(NULL));
12     i = rand100();
13     printf("%d\n", i);
14     printf("%d\n", rand100());
15     exit(EXIT_SUCCESS);
16 }
17
18 /* Return a random int in the range 1 to 100 inclusive. */
19
20 int rand100(void)
21 {
22     return rand() % 100 + 1;
23 }
```

```
63
24
```

```
 1 #include <stdio.h>
 2 #include <stdlib.h>
 3 #include <time.h>
 4
 5 int mislead(void);
 6
 7 main()
 8 {
 9     int code;        /* error code */
10
11     srand(time(NULL));
12     code = mislead();
13     if (code > 1) {
14         printf("This is a very severe error.\n");
15     }
16     exit(EXIT_FAILURE);
17 }
18
19 /* Print error message, return error code number. */
20 int mislead(void)
21 {
```

```
22      int r = rand() % 3;
23
24      if (r == 0) {
25          printf("core dumped\n");
26      } else if (r == 1) {
27          printf("memory fault\n");
28      } else {
29          printf("segmentation violation\n");
30      }
31
32      printf("error code %d\n", r);
33      return r;
34  }
```

```
segmentation violation
error code 2
This is a very severe error.
```

Ignore the return value of printf

You can call a function and ignore its return value. That is what we've been doing all along with **printf**: it returns the number of characters that were output (K&R p. 153; King p. 487). We could have written

```
 1  #include <stdio.h>
 2  #include <stdlib.h>
 3
 4  main()
 5  {
 6      int count = printf("hello\n");
 7
 8      if (count == 6) {
 9          exit(EXIT_SUCCESS);
10      } else {
11          exit(EXIT_FAILURE);
12      }
13  }
```

You can change lines 6−12 to

```
 8      exit(printf("hello\n") == 6 ? EXIT_SUCCESS : EXIT_FAILURE);
```

Ignore the return value of scanf: K&R p. 157; King p. 492

```
 1  #include <stdio.h>
 2  #include <stdlib.h>
 3
 4  main()
 5  {
 6      int i;
 7
 8      while (scanf("%d", &i) == 1) {
 9          if (i % 2 == 0) {
10              printf("%d is even.\n", i);
11          } else {
```

```
12              printf("%d is odd.\n", i);
13          }
14      }
15
16      printf("That's all, folks!\n");
17  }
```

```
2
2 is even
3
3 is odd
control-d
That's all, folks!
```

Pass an argument to a function (pass-by-value): K&R pp. 24−27; King p. 165

```
 1  #include <stdio.h>
 2  #include <stdlib.h>
 3
 4  main()
 5  {
 6      int i;
 7
 8      /* Draw a horizontal line of 10 dashes. */
 9      for (i = 0; i < 10; ++i) {
10          putchar('-');
11      }
12      putchar('\n');
13
14      /* Draw a horizontal line of 20 '-'s. */
15      for (i = 0; i < 20; ++i) {
16          putchar('-');
17      }
18      putchar('\n');
19
20      exit(EXIT_SUCCESS);
21  }
```

```
----------
--------------------
```

```
22  #include <stdio.h>
23  #include <stdlib.h>
24
25  void line(int n);                   /* can omit the "n" from this line: K&R p. 26;
26                                         King pp. 164−165 */
27
28  main()
29  {
30      line(10);
31      line(20);
32      exit(EXIT_SUCCESS);
33  }
```

```
34
35 /* Draw a horizontal line of n dashes. */
36
37 void line(int n)
38 {
39     int i;
40
41     for (i = 0; i < n; ++i) {
42         putchar('-');
43     }
44     putchar('\n');
45 }
```

The parenthesized expressions in lines 8 and 9 above do not have to be constants. They can also be variables or more complicated expressions.

▼ Homework 7.3: use a parameter as a conveniently initialized local variable (required reading)

```
 1 /* Draw a horizontal line of n dashes. */
 2
 3 void line(int n)
 4 {
 5     for (; n > 0; --n) {
 6         putchar('-');
 7     }
 8     putchar('\n');
 9 }

10 /* Draw a horizontal line of n dashes. */
11
12 void line(int n)
13 {
14     while (n-- > 0) {
15         putchar('-');
16     }
17     putchar('\n');
18 }
```

Compare the **power** function K&R p. 25; King p. 165 with the **power** function K&R p. 27; King p. 166. Even though one counts up and the other counts down, they both return the same result. The second version, however, has fewer variables. It does not damage the value of the variable used as its second argument in the calling function (i.e., the variable **i** at the bottom of K&R p. 24).

▲

A function with two arguments

```
 1 #include <stdio.h>
 2 #include <stdlib.h>
 3
 4 main()
 5 {
 6     int i;
 7
 8     /* Draw a horizontal line of 10 dashes. */
 9     for (i = 0; i < 10; ++i) {
10         putchar('-');
```

```
    }
    putchar('\n');

    /* Draw a horizontal line of 20 equal signs. */
    for (i = 0; i < 20; ++i) {
        putchar('=');
    }
    putchar('\n');

    exit(EXIT_SUCCESS);
}
```

```
----------
====================
```

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

void line(int n, char c);

main()
{
    line(10, '-');       /* Draw a horizontal line of 10 dashes. */
    line(20, '=');       /* Draw a horizontal line of 20 equal signs. */

    exit(EXIT_SUCCESS);
}

/* Draw a horizontal line made of n copies of character c. */

void line(int n, char c)
{
    for (; n > 0; --n) {
        putchar(c);
    }
    putchar('\n');
}
```

```
/* Draw a horizontal line made of n copies of character c. */

void line(int n, char c)
{
    if (n < 0) {
        printf("length must be non-negative\n");
        return;
    }

    for (; n > 0; --n) {
        putchar(c);
    }
    putchar('\n');
}
```

A useful function with two arguments

Before using this function, make sure that no one has created a **`#define`** named **`max`** or **`min`**. Change it to return the average of **`min`** and **`max`** after three incorrect inputs.

```
 1 int getint(int max, int min)
 2 {
 3     int n;
 4
 5     printf("Please type a number between %d and %d inclusive, ", min, max);
 6     printf("and press RETURN: ");
 7
 8     for (;;) {
 9         scanf("%d", &n);
10         if (min <= n && n <= max) {
11             return n;
12         }
13         printf("Sorry, the number must be between %d and %d inclusive.\n",
14             min, max);
15         printf("Please try again: ");
16     }
17 }
```

▼ Homework 7.4: consolidate using a function

Eliminate repetitious code from the following program by creating a function whose declaration is

```
void printint(int n, int base);
```

that will print the integer **`n`** in base **`base`**, provided that $2 \le$ **`base`** ≤ 10.

```
 1 #include <stdio.h>
 2 #include <stdlib.h>
 3
 4 main()
 5 {
 6     int n;
 7     int i;
 8
 9     printf("Press RETURN after you type each number.\n");
10
11     while (scanf("%d", &n) == 1) {
12
13         /* Print n in binary.  First find where to begin. */
14         for (i = 1; i*2 <= n; i *= 2) {
15         }
16
17         for (; i > 0; i /= 2) {
18             printf("%d", n / i % 2);     /* Handout 3, p. 17, third line 14 */
19         }
20         printf("\n");
21
22         /* Now print n in octal. */
23         for (i = 1; i*8 <= n; i *= 8) {
24         }
25
26         for (; i > 0; i /= 8) {
```

```
27              printf("%d", n / i % 8);
28          }
29          printf("\n");
30      }
31
32      exit(EXIT_SUCCESS);
33  }
```

▲

Pass-by-value

```
 1  #include <stdio.h>
 2  #include <stdlib.h>
 3
 4  void f(int n);
 5
 6  main()
 7  {
 8      int i = 10;
 9      f(i);
10      printf("%d\n", i);       /* It still prints 10. */
11      exit(EXIT_SUCCESS);
12  }
13
14  void f(int n)
15  {
16      printf("%d\n", n);       /* It prints 10. */
17      ++n;
18      printf("%d\n", n);       /* It prints 11. */
19  }
```

```
10
11
10
```

Pass-by-reference: K&R pp. 27−28, 95−96; King p. 211

Don't say **++p**—then **p** would no longer point anywhere useful.

```
 1  #include <stdio.h>
 2  #include <stdlib.h>
 3
 4  void f(int *p);
 5
 6  main()
 7  {
 8      int i = 10;
 9
10      f(&i);                   /* Requires &, as does scanf. */
11      printf("%d\n", i);       /* Now it will print 11. */
12      exit(EXIT_SUCCESS);
13  }
14
15  void f(int *p)
16  {
```

```
17      printf("%d\n", *p);      /* It prints 10. */
18      ++*p;                    /* It means *p = *p + 1; */
19      printf("%d\n", *p);      /* It prints 11. */
20  }
```

```
10
11
11
```

▼ **Homework 7.5: fix the bug**

The arithmetic in the following program is correct, but the communication between functions is wrong. Change the pass-by-value to pass-by-reference. Let the return type of **`f`** remain **`void`**. Use a **`const int N`** instead writing 100 everywhere. The number **`100`** must be written in only one place (the initialization of the **`const int`**), and the number **`50`** should be written as **`n/2`**. Change the comments too.

```
 1  #include <stdio.h>
 2  #include <stdlib.h>
 3
 4  void f(int j);
 5
 6  main()
 7  {
 8      int i;
 9
10      printf("Please type a number and press RETURN.\n");
11      scanf("%d", &i);
12
13      f(i);
14      printf("Rounded to the nearest %d, the number is %d\n", 100, i);
15
16      exit(EXIT_SUCCESS);
17  }
18
19  /* Round n to the nearest 100.  Round n up if tied. */
20
21  void f(int j)
22  {
23      int remainder = j % 100;
24
25      if (remainder < 50) {
26          j -= remainder;
27      } else {
28          j += (100 - remainder);                /* parentheses unnecessary */
29      }
30  }
```

▲

▼ **Homework 7.6: write a string function**

Write one of the string functions on K&R pp. 249−250; King pp. 627−631, and a **`main`** function to call it. For example,

▲

```
1  /* Excerpt from /usr/include/stdlib.h: from now on, the word size_t
```

```
 2 stands for unsigned int.  On your machine, it may stand for another type. */
 3
 4 typef unsigned int size_t;
```

```
 1 #include <stdio.h>
 2 #include <stdlib.h>
 3 #define N 256
 4
 5 char *strcpy(char *dest, char *source);
 6
 7 main()
 8 {
 9     char source[N];
10     char dest[N];
11
12     printf("Type a line and press RETURN.\n");
13     gets(source);
14     strcpy(dest, source);
15     printf("%s\n", dest);
16     exit(EXIT_SUCCESS);
17 }
18
19 char *strcpy(char *dest, char *source)/* K&R pp. 104-106; King p. 252 */
20 {
21     char *save = dest;
22
23     while ((*dest++ = *source++) != '\0') {
24     }
25
26     return save;
27 }
```

Last hired, first fired

A *stack* is a classic information storage and retrieval system. Accountants call it a ‘‘lifo’’ list: last in, first out. See K&R p. 77; King p. 187.

```
 1 #include <stdio.h>
 2 #include <stdlib.h>
 3 #define MAXVAL 100
 4
 5 main()
 6 {
 7     int val[MAXVAL]; /* the stack itself.  Your machine may need "long". */
 8     int sp = 0;      /* stack pointer */
 9
10     int ss;          /* Your machine may need "long". */
11
12     printf("To hire a person, type their social security number.\n");
13     printf("To fire the most recently hired person, type a zero.\n");
14     printf("To exit, type a negative number.\n");
15
16     for (;;) {
17         scanf("%d", &ss);
```

```
18          if (ss < 0) {
19              break;                              /* K&R pp. 64-65; King pp. 97–98 */
20          }
21
22          if (ss > 0) {                           /* hire */
23              if (sp < MAXVAL) {
24                  val[sp] = ss;
25                  sp++;
26              } else {
27                  printf("Sorry, there are already %d employees.\n", MAXVAL);
28              }
29          } else {                                /* fire */
30              if (sp > 0) {
31                  --sp;
32                  printf("Firing number %d\n", val[sp]);
33              } else {
34                  printf("Sorry, there's no one left to fire.\n");
35              }
36          }
37      }
38
39      exit(EXIT_SUCCESS);
40  }
```

Package the stack operations in functions

Make the logic in the **`main`** function simpler by banishing the stack machinery to separate functions.

```
 1  #include <stdio.h>
 2  #include <stdlib.h>
 3  #define MAXVAL 100
 4
 5  void push(int n);
 6  int pop(void);
 7
 8  int val[MAXVAL];
 9  int sp = 0;          /* stack pointer */
10
11  main()
12  {
13      int ss;          /* You may need "long" on your machine. */
14
15      printf("To hire a person, type their social security number.\n");
16      printf("To fire the most recently hired person, type a zero.\n");
17      printf("To exit, type a negative number.\n");
18
19      for (;;) {
20          scanf("%d", &ss);
21          if (ss < 0) {
22              break;
23          }
24
25          if (ss > 0) {                           /* hire */
26              push(ss);
27          } else {                                /* fire */
```

```
28              printf("Firing number %d\n", pop());
29          }
30      }
31
32      exit(EXIT_SUCCESS);
33  }
34
35  /* Push a number onto the top of the stack. */
36  void push(int n)
37  {
38      if (sp < MAXVAL) {
39          val[sp] = n;
40          sp++;
41      } else {                                    /* overflow */
42          printf("stack already contains %d values\n", MAXVAL);
43      }
44  }
45
46  /* Pop a number off the top of the stack. */
47  int pop (void)
48  {
49      if (sp > 0) {
50          --sp;
51          return val[sp];
52      } else {                                    /* underflow */
53          printf("stack is already empty\n");
54          return 0;
55      }
56  }
```

▼ **Homework 7.7: convert to pointer notation**

In the above program, combine lines 39−40 into a single line. Combine lines 50−51 into a single line. Then change

```
 8 int sp = 0;
```

to

```
 8 int *sp = val;
```

▲

Rudimentary check for incorrectly paired {curly braces}

```
 1 #include <stdio.h>
 2 #include <stdlib.h>
 3
 4 main()
 5 {
 6     int c;
 7     int count = 0;
 8
 9     while ((c = getchar()) != EOF) {
10         if (c == '{') {
11             ++count;
12         } else if (c == '}') {
```

```
13              --count;
14          }
15      }
16
17      printf("%d\n", count);
18      if (count == 0) {
19          exit(EXIT_SUCCESS);
20      } else {
21          exit(EXIT_FAILURE);
22      }
23  }
```

You can change lines 18−22 to

```
18      exit(count == 0 ? EXIT_SUCCESS : EXIT_FAILURE);
```

```
1$ gcc -o rudy rudy.c          Create an executable file called rudy
2$ rudy < rudy.c               Feed it a copy of its own source code.
0                              It prints the number zero.
3$ echo $status                EXIT_SUCCESS is zero on my machine
0
```

▼ Homework 7.8: Check for incorrectly paired curly braces

Only the **main** function of the following program (called **pair.c**) is shown below. You must add the **push** and **pop** functions you wrote in the previous Homework. Remove the error message from **pop**. For simplicity, put the whole program in one **.c** file. The stack will hold the line number of each **'{'** encountered in the input.

```
 1 /* Input a .c file and report the line number of every right character
 2 that has no matching left character.  After reading all the input,
 3 report the line number of every left character that had no matching
 4 right character. */
 5
 6 #include <stdio.h>
 7 #include <stdlib.h>
 8
 9 #define MAXVAL 100
10 int val[MAXVAL];
11 int *sp = val;
12
13 main()
14 {
15      int c;                          /* each char read from input */
16      int lineno = 1;                 /* current line number: see K&R p. 19  */
17      int status = EXIT_SUCCESS;      /* innocent until proven guilty */
18
19      /* Other pairs of choices might be '(' ')' or '[' ']'. */
20      const char left = '{';
21      const char right = '}';
22
23      while ((c = getchar()) != EOF) {
24          if (c == '\n') {
25              ++lineno;
26          } else if (c == left) {
27              push(lineno);
```



```
28          } else if (c == right && pop() == 0) {
29              printf("%c without previous %c on line %d\n", right, left, lineno);
30              status = EXIT_FAILURE;
31          }
32      }
33
34      /* After reading all the input, print out the stack.
35      If there were no errors, the stack will be empty. */
36      while ((lineno = pop()) != 0) {
37          printf("%c without following %c on line %d\n", left, right, lineno);
38          status = EXIT_FAILURE;
39      }
40
41      exit(status);
42  }
```

▲

Two ways to pass a one-dimensional array to a function

A function may receive an array as an argument. What actually gets passed to the function is the address of the first element of the array. Therefore the function can declare the argument to be either an array or a pointer. ‘‘[W]e prefer the latter,’’ remark the authors magisterially on K&R p. 100.

```
 1  #include <stdio.h>
 2  #include <stdlib.h>
 3
 4  void f(char a[]);
 5  void g(char *p);
 6
 7  main()
 8  {
 9      char word[] = "hello";       /* an array of six characters */
10
11      f(word);
12      g(word);
13
14      exit(EXIT_SUCCESS);
15  }
16
17  void f(char a[])             /* Array notation. */
18  {
19      printf("%s\n", a);       /* Print the entire string. */
20      printf("%c\n", a[0]);    /* Print only 1st char; couldn't say *a instead. */
21  }
22
23  void g(char *p)              /* Pointer notation. */
24  {
25      printf("%s\n", p);       /* Print the entire string. */
26      printf("%c\n", p[0]);    /* Print only 1st char; could say *p instead. */
27  }
```

Using either of these notations, the only thing that the function receives is the address of the first element of the array. The function has no way of knowing how many array elements there are. Therefore you should not put a number in the square brackets in lines 4 and 17. The following call to **`scanf`**


```
#define N   10
    char s[N];
    scanf("%s", s);      /* Input one word. */
```

will overwrite memory beyond the end of the array if the user inputs a word of **N** or more characters.

Pass two-dimensional arrays of various sizes to a function

If you have provided an initial value for each element, you can remove the first dimension (the empty brackets in line8 and 13) but not the other dimensions.

You can remove the **{**curly braces**}** in lines 9−10 and 14−17, but not the ones in lines 8, 11, 13, 18.

See Stroustrup, *The C++ Programming Language,* pp. 838−840, for examples that are valid in C as well as C++.

—On the Web at
http://i5.nyu.edu/~mm64/x52.9232/src/multisoft.c

```
 1 #include <stdio.h>
 2 #include <stdlib.h>
 3
 4 void f(int *p, int nrows, int ncols);
 5
 6 main()
 7 {
 8     int a1[][3] = {
 9         {0, 1, 2},
10         {3, 4, 5}
11     };
12 #define A1_ROWS (sizeof a1 / sizeof a1[0])
13
14     int a2[][5] = {
15         { 2,  3,  5,  7, 11},
16         {13, 17, 19, 23, 29},
17         {31, 37, 41, 43, 47},
18         {51, 53, 57, 61, 67}
19     };
20 #define A2_ROWS (sizeof a2 / sizeof a2[0])
21
22     f(&a1[0][0], A1_ROWS, 3);
23     printf("\n");
24     f(&a2[0][0], A2_ROWS, 5);
25 }
26
27 void f(int *p, int nrows, int ncols)
28 {
29     int row;
30     int col;
31
32     for (row = 0; row < nrows; ++row) {
33         printf("row %2d: ", row);
34         for (col = 0; col < ncols; ++col) {
35             printf("%2d", p[ncols * row + col]);
36             if (col < ncols - 1) {
37                 printf(" ");
38             }
39         }
```

```
40          printf("\n");
41      }
42  }
```

```
row  0:  0  1  2
row  1:  3  4  5

row  0:  2  3  5  7 11
row  1: 13 17 19 23 29
row  2: 31 37 41 43 47
row  3: 51 53 57 61 67
```

Add two macros more macros, **A1_COLS** and **A2_COLS**.

A C array is always stored in row-major order (‘‘last subscript moves fastest’’). In other words, the first two **int**’s at the start of the array are **a1[0][0]** and **a1[0][1]**. Had the array been stored in column-major order (‘‘first subscript moves fastest’’, so that the first two **int**’s were **a1[0][0]** and **a1[1][0]**), line 35 would have to be

```
35              printf("%2d", p[nrows * col + row]);
```

See K&R p. 217.

You could say

```
26  #define P(r, c)  (p[ncols * (r) + (c)])
43  #undef P
```

and then you could change line 35 to

```
35              printf("%2d", P(row, col));
```

□