APPENDIX A

ANSI Standard Header Files

As you have learned in the past 24 hours, the C standard library comes with a set of include files called *header files*. These header files contain the declarations for the C library functions and macros, as well as relevant data types. Whenever a C function is invoked, the header file(s) with which the C function is associated has to be included in your programs.

The following are the ANSI standard header files:

File	*Description*
`assert.h`	Contains diagnostic functions.
`ctype.h`	Contains character testing and mapping functions.
`errno.h`	Contains constants for error processing.
`float.h`	Contains constants for floating-point values.
`limits.h`	Contains implementation-dependent values.

continues

File	*Description*
`locale.h`	Contains the `setlocale()` function, and is used to set locale parameters.
`math.h`	Contains mathematics functions.
`setjmp.h`	Contains the `setjmp()` and `longjmp()` functions, and is used to bypass the normal function call and return discipline.
`signal.h`	Contains signal-handling functions.
`stdarg.h`	Contains functions and macros for implementing functions that accept a variable number of arguments.
`stddef.h`	Contains definitions for the `ptrdiff_t`, `size_t`, `NULL`, and `errno` macros.
`stdio.h`	Contains input and output functions.
`stdlib.h`	Contains general utility functions.
`string.h`	Contains functions that are used to manipulate strings.
`time.h`	Contains functions for manipulating time.

If the C compiler on your machine is not 100 percent ANSI-conformable, some of the ANSI header files might not be available with the compiler.

APPENDIX B

Answers to Quiz Questions and Exercises

Hour 1, "Taking the First Step"

Quiz

1. The lowest language mentioned in this hour that a computer can understand directly is the machine language—that is, the binary code. On the other hand, the highest language is the human language, such as Chinese, English, French, and so on. Most high-level programming languages, such as C, Java, and Perl, are close to the human language.
2. A computer cannot directly understand a program written in C. You have to compile the program and translate it into binary code so that the computer can read it.
3. Yes. That's the beauty of the C language; you can write a program in C and save it into a library file. Later, you can invoke the program in another C program by including the library file.

4. We need the ANSI standard for C to guarantee the portability of the programs written in C. Most C compiler vendors support the ANSI standard. If you write your program by following the rules set up by the ANSI standard, you can port your program to any machine by simply recompiling your program with a compiler that supports those machines.

Hour 2, "Writing Your First C Program"

Quiz

1. No. Actually, the C preprocessor will filter out all comments you put into your program before the compiler can see them. Comments are written for you or other programmers who look at your program.
2. An `.obj` file is created after a program is compiled by the C compiler. You still need a linker to link all `.obj` files and other library files together to make the final executable file.
3. No, the `exit()` function doesn't return any values. However, the `return` statement does. In the `main()` function, if the `return` statement returns a value of `0`, it indicates to the operating system that the program has terminated normally; otherwise, an error occurs.
4. A file that is required by the `#include` directive and ends with the extension `.h` is called a *header file* in C. Later in this book, you'll learn that a header file contains the data or function declarations.

Exercises

1. No. The angle brackets (< and >) in the `#include <stdio.h>` expression ask the C preprocessor to look for a header file in a directory other than the current one. On the other hand, the `#include "stdio.h"` expression tells the C preprocessor to check the current directory first for the header file `stdio.h`, and then look for the header file in another directory.
2. The following is one possible solution:

```
/* 02A02.c */
#include <stdio.h>

main()
{
   printf ("It's fun to write my own program in C.\n");
   return 0;
}
```

OUTPUT The output of the program is:

```
It's fun to write my own program in C.
```

3. The following is one possible solution:

```
/* 02A03.c */
#include <stdio.h>

main()
{
   printf ("Howdy, neighbor!\nThis is my first C program.\n");
   return 0;
}
```

OUTPUT The output of the program is:

```
Howdy, neighbor!
This is my first C program.
```

4. The warning message I get when I try to compile the program is that the `main()` function should return a value of integer because, by default, the `main()` function returns an integer. Because the `exit()` function doesn't return any values, you can replace `exit()` with the `return` statement.
5. I got two error (warning) messages on my machine. The first one is `'printf' undefined`; the second one is `'main' : 'void' function returning a value`. To fix the first error, the header file, `stdio.h`, has to be included first before the `printf()` function can be called from the `main()` function; otherwise, you'll get an error message during the linking stage. To fix the second error, you can remove the `void` keyword from the code.

Hour 3, "Learning the Structure of a C Program"

Quiz

1. Yes. Both `74` and `571` are constants in C.
2. Yes. Both `x = 571 + 1` and `x = 12 + y` are expressions.
3. `2methods`, `*start_function`, and `.End_Exe` are not valid function names.
4. No. `2 + 5 * 2` is equivalent to `2 + 10`, which gives `12`; `(2 + 5) * 2` is equivalent to `7 * 2`, which produces `14`.
5. Yes. Both `7 % 2` and `4 % 3` produce `1`.

Exercises

1. The following is one possible solution:

```
{
  x = 3;
  y = 5 + x;
}
```

2. The function name, `3integer_add`, is illegal in C.
3. The second statement inside the function needs a semicolon at the end of the statement.
4. The following are two possible solutions:

```
/* Method 1: a C function */
int MyFunction( int x, int y)
{
   int result;
   result = x * y;
   return result;
}
```

or

```
/* Method 2: a C function */
int MyFunction( int x, int y)
{
   return (x * y);
}
```

5. The following is one possible solution:

```
/* 03A05.c */
#include <stdio.h>

int integer_multiply( int x, int y )
{
   int result;
   result = x * y;
   return result;
}

int main()
{
   int sum;

   sum = integer_multiply(3, 5);
   printf("The multiplication of 3 and 5 is %d\n", sum);
   return 0;
}
```

Hour 4, "Understanding Data Types and Keywords"

Quiz

1. Yes. Both `134/100` and `17/10` give the same result of `1`.
2. Yes. The results of both `3000 + 1.0` and `3000/1.0` are floating-point values.
3. In scientific notation, we have the following expressions:
   - `3.5e3`
   - `3.5e-3`
   - `-3.5e-3`
4. Among the four names, `7th_calculation` and `Tom's_method` are not valid names in C.

Exercises

1. The following is one possible solution:

```
/* 04A01.c */
#include <stdio.h>

main()
{
   char c1;
   char c2;

   c1 = 'Z';
   c2 = 'z';
   printf("The numeric value of Z: %d.\n", c1);
   printf("The numeric value of z: %d.\n", c2);
   return 0;
}
```

OUTPUT The output of the program is:

```
The numeric value of Z: 90.
The numeric value of z: 122.
```

2. The following is one possible solution:

```
/* 04A02.c */
#include <stdio.h>

main()
{
   char c1;
```

```
   char c2;

   c1 = 72;
   c2 = 104;
   printf("The character of 72 is: %c\n", c1);
   printf("The character of 104 is: %c\n", c2);
   return 0;
}
```

OUTPUT The output of the program is:

```
The character of 72 is: H
The character of 104 is: h
```

3. No. `72368` is beyond the range of the `int` data type of 16 bits. If you assign a value that is too large for the data type, the resulting value will wrap around and the result will be incorrect.

4. The following is one possible solution:

```
/* 04A04.c */
#include <stdio.h>

main()
{
   double dbl_num;;

   dbl_num = 123.456;
   printf("The floating-point format of 123.456 is: %f\n",
           dbl_num);
   printf("The scientific notation format of 123.456 is: %e\n",
           dbl_num);

   return 0;
}
```

OUTPUT The output of the program from my machine is:

```
The floating-point format of 123.456 is: 123.456000
The scientific notation format of 123.456 is: 1.234560e+002
```

5. The following is one possible solution:

```
/* 04A05.c */
#include <stdio.h>

main()
{
   char ch;

   ch = '\n';
   printf("The numeric value of newline is: %d\n", ch);

   return 0;
}
```

OUTPUT The output of the program is:

```
The numeric value of newline is: 10
```

Hour 5, "Handling Standard Input and Output"

Quiz

1. Yes. By prefixing the minimum field specifier with the minus sign `-`.
2. The main difference between `putc()` and `putchar()` is that `putc()` requires the user to specify the file stream. For `putchar()`, the user doesn't need to do so because the standard output (`stdout`) is used as the file stream.
3. The `getchar()` function returns a value of the `int` data type.
4. Within the `%10.3f` expression, `10` is the value of the minimum field width specifier; `.3` is called the precision specifier.

Exercises

1. The following is one possible solution:

```
/* 05A01.c */
#include <stdio.h>

main()
{
   char c1, c2, c3;

   c1 = 'B';
   c2 = 'y';
   c3 = 'e';

   /* Method I */
   printf("%c%c%c\n", c1, c2, c3);

   /* Method II */
   putchar(c1);
   putchar(c2);
   putchar(c3);

   return 0;
}
```

2. The following is one possible solution:

```
/* 05A02.c */
#include <stdio.h>
```

```
main()
{
   int x;
   double y;

   x = 123;
   y = 123.456;
   printf("x: %-3d\n", x);
   printf("y: %-6.3f\n", y);

   return 0;
}
```

OUTPUT

The output of the program is:

```
x: 123
y: 123.456
```

3. The following is one possible solution:

```
/* 05A03.c */
#include <stdio.h>

main()
{
   int num1, num2, num3;

   num1 = 15;
   num2 = 150;
   num3 = 1500;
   printf("The hex format of 15 is: 0x%04X\n", num1);
   printf("The hex format of 150 is: 0x%04X\n", num2);
   printf("The hex format of 1500 is: 0x%04X\n", num3);

   return 0;
}
```

OUTPUT

The output of the program is:

```
The hex format of 15 is: 0x000F
The hex format of 150 is: 0x0096
The hex format of 1500 is: 0x05DC
```

4. The following is one possible solution:

```
/* 05A04.c */
#include <stdio.h>

main()
{
   int ch;

   printf("Enter a character:\n");
   ch = getchar();
```

```
    putchar(ch);

    return 0;
}
```

5. You will probably get two error (warning) messages; one stating that `getchar()` is undefined and another saying that `putchar()` is undefined. The reason is that the header file, `stdio.h`, is missing in the code.

Hour 6, "Manipulating Data"

Quiz

1. The = operator is an assignment operator that assigns the value of the operand on the right side of the operator to the one on the left side. On the other hand, == is one of the relational operators; it just tests the values of two operands on both sides and finds out whether they are equal to each other.
2. In the `x + - y - - z` expression, the first and third minus signs are unary minus operators; the second minus sign is a subtraction operator.
3. `15/4` evaluates to `3`. `(float)15/4` evaluates to `3.750000`.
4. No. The `y *= x + 5` expression is actually equal to the `y = y * (x + 5)` expression.

Exercises

1. The following is one possible solution:

```
/* 06A01.c */
#include <stdio.h>

main()
{
    int x, y;

    x = 1;
    y = 3;
    x += y;
    printf("The result of x += y is: %d\n", x);

    x = 1;
    y = 3;
    x += -y;
    printf("The result of x += -y is: %d\n", x);

    x = 1;
    y = 3;
```

```
   x -= y;
   printf("The result of x -= y is: %d\n", x);

   x = 1;
   y = 3;
   x -= -y;
   printf("The result of x -= -y is: %d\n", x);

   x = 1;
   y = 3;
   x *= y;
   printf("The result of x *= y is: %d\n", x);

   x = 1;
   y = 3;
   x *= -y;
   printf("The result of x *= -y is: %d\n", x);

   return 0;
}
```

OUTPUT The output of the program is:

```
The result of x += y is: 4
The result of x += -y is: -2
The result of x -= y is: -2
The result of x -= -y is: 4
The result of x *= y is: 3
The result of x *= -y is: -3
```

2. The value of `z` is `1` (one), after the expression `z=x*y==18` expression is evaluated.
3. The following is one possible solution:

```
/* 06A03.c */
#include <stdio.h>

main()
{
   int x;

   x = 1;
   printf("x++ produces:   %d\n", x++);
   printf("Now x contains: %d\n", x);

   return 0;
}
```

OUTPUT The output of the program is:

```
x++ produces:   1
Now x contains: 2
```

4. The following is one possible solution:

```
/* 06A04.c */
#include <stdio.h>

main()
{
   int x;

   x = 1;
   printf("x = x++ produces: %d\n", x = x++);
   printf("Now x contains:   %d\n", x);

   return 0;
}
```

   I get `1` and `1` from the two `printf()` calls in this program. The reason is that, in the `x = x++` expression, the original value of `x` is copied into a temporary location first, and then `x` is incremented by 1. Last, the value saved in the temporary location is assigned back to `x`. That's why the final value saved in `x` is still `1`.

5. The program incorrectly uses an assignment operator `=`, instead of an "equal to" relational operator (`==`).

Hour 7, "Working with Loops"

Quiz

1. No.
2. Yes. The `do-while` loop prints out the character `d`, whose numeric value is `100`.
3. Yes. Both `for` loops iterate `8` times.
4. Yes.

Exercises

1. The first `for` loop contains a statement:

```
printf("%d  +  %d  =  %d\n", i, j, i+j);
```

   But the second `for` loop has a semicolon right after the for statement. This is a null statement—a semicolon by itself, which is a statement that does nothing.

2. The following is one possible solution:

```
/* 07A02.c */
#include <stdio.h>

main()
{
```

```
  int i, j;

  for (i=0, j=1; i<8; i++, j++)
     printf("%d  +  %d  =  %d\n", i, j, i+j);

  printf("\n");
  for (i=0, j=1; i<8; i++, j++);
     printf("%d  +  %d  =  %d\n", i, j, i+j);

  return 0;
}
```

3. The following is one possible solution:

```
/* 07A03.c */
#include <stdio.h>

main()
{
   int c;

   printf("Enter a character:\n(enter K to exit)\n");
   c = ' ';

   while( c != 'K' ) {
      c = getc(stdin);
      putchar(c);
   }
   printf("\nOut of the for loop. Bye!\n");

   return 0;
}
```

4. The following is one possible solution:

```
/* 07A04.c: Use a for loop */
#include <stdio.h>

main()
{
   int i;

   i = 65;
   for (i=65; i<72; i++){
      printf("The numeric value of %c is %d.\n", i, i);
   }

   return 0;
}
```

5. The following is one possible solution:

```
/* 07A05.c */
#include <stdio.h>

main()
{
   int i, j;

   i = 1;
   while (i<=3) {   /* outer loop */
      printf("The start of iteration %d of the outer loop.\n", i);
      j = 1;
      do{   /* inner loop */
         printf("    Iteration %d of the inner loop.\n", j);
         j++;
      } while (j<=4);
      i++;
      printf("The end of iteration %d of the outer loop.\n", i);
   }

   return 0;
}
```


Hour 8, "Using Conditional Operators"

Quiz

1. The `(x=1)&&(y=10)` expression returns `1`; `(x=1)&(y=10)` returns `0`.
2. In the `!y ? x == z : y` expression, `!y` produces `0`, thus the value of the third operand `y` is taken as the value of the expression. That is, the expression evaluates to `1`.
3. `1100111111000110` and `0011000000111001`.
4. The `(x%2==0)¦¦(x%3==0)` expression yields `1`, and the `(x%2==0)&&(x%3==0)` expression evaluates to `0`.
5. Yes. `8 >> 3` is equivalent to $8/2^3$. `1 << 3` is equivalent to 2^3.

Exercises

1. `~x` yields `0x1000` because `~0xEFFF` is equivalent to `~0111111111111111` (in binary), which produces `1000000000000000` (in binary), (that is, `0x1000` in hex format). Likewise, `~y` evaluates to `0xEFFF` because `~0x1000` is equivalent to `~1000000000000000` (in binary), which yields `0111111111111111` (in binary) (that is, `0xEFFF` in hex format).

2. The following is one possible solution:

```
/* 08A02.c */
#include <stdio.h>

int main()
{
   int x, y;

   x = 0xEFFF;
   y = 0x1000;

   printf("!x yields: %d (i.e., %u)\n", !x, !x);
   printf("!y yields: %d (i.e., %u)\n", !y, !y);

   return 0;
}
```

   The output of the program is:

```
!x yields: 0 (i.e., 0)
!y yields: 0 (i.e., 0)
```

3. The following is one possible solution:

```
/* 08A03.c */
#include <stdio.h>

int main()
{
   int x, y;

   x = 123;
   y = 4;
   printf("x << y yields: %d\n", x << y);
   printf("x >> y yields: %d\n", x >> y);

   return 0;
}
```

   [ic:output]The output of the program is:

```
x << y yields: 1968
x >> y yields: 7
```

4. The following is one possible solution:

```
/* 08A04.c */
#include <stdio.h>

int main()
{
   printf("0xFFFF ^ 0x8888 yields: 0x%X\n",
```

```
            0xFFFF ^ 0x8888);
   printf("0xABCD & 0x4567 yields: 0x%X\n",
            0xABCD & 0x4567);
   printf("0xDCBA ¦ 0x1234 yields: 0x%X\n",
            0xDCBA ¦ 0x1234);

   return 0;
}
```

OUTPUT The output of the program is:

```
0xFFFF ^ 0x8888 yields: 0x7777
0xABCD & 0x4567 yields: 0x145
0xDCBA ¦ 0x1234 yields: 0xDEBE
```

5. The following is one possible solution:

```
/* 08A05.c */
#include <stdio.h>

main()
{
   int x;

   printf("Enter a character:\n(enter q to exit)\n");
   for ( x=' '; x != 'q' ? 1 : 0; ) {
      x = getc(stdin);
      putchar(x);
   }
   printf("\nOut of the for loop. Bye!\n");

   return 0;
}
```


Hour 9, "Working with Data Modifiers and Math Functions"

Quiz

1. No. `x` contains a negative number that is not the same as the number contained by the `unsigned int` variable `y`.
2. You can use the `long` modifier, which increases the range of values that the `int` can hold. Or, if you are storing a positive number, you can also use the `unsigned` modifier to store a larger value.
3. `%lu`.
4. The header file is `math.h`.

Exercises

1. The following is one possible solution:

```
/* 09A01 */
#include <stdio.h>

main()
{
   int x;
   unsigned int y;

   x = 0xAB78;
   y = 0xAB78;

   printf("The decimal value of x is %d.\n", x);
   printf("The decimal value of y is %u.\n", y);

   return 0;
}
```

OUTPUT The output of the program from my machine is:

```
The decimal value of x is -21640.
The decimal value of y is 43896.
```

2. The following is one possible solution:

```
/* 09A02 */
#include <stdio.h>

main()
{
   printf("The size of short int is %d.\n",
          sizeof(short int));
   printf("The size of long int is %d.\n",
          sizeof(long int));
   printf("The size of long double is %d.\n",
          sizeof(long double));

   return 0;
}
```

3. The following is one possible solution:

```
/* 09A03 */
#include <stdio.h>

main()
{
   int x, y;
   long int result;

   x = 7000;
```

```
   y = 12000;
   result = x * y;
   printf("x * y == %lu.\n", result);

   return 0;
}
```

OUTPUT The output of the program from my machine is:

```
x * y == 84000000.
```

4. The following is one possible solution:

```
/* 09A04 */
#include <stdio.h>

main()
{
   int x;

   x = -23456;
   printf("The hex value of x is  0x%X.\n", x);

   return 0;
}
```

OUTPUT The output of the program from my machine is:

```
The hex value of x is  0xA460.
```

5. The following is one possible solution:

```
/* 09A05.c */
#include <stdio.h>
#include <math.h>

main()
{
   double x;

   x = 30.0;                /* 45 degrees */
   x *= 3.141593 / 180.0;    /* convert to radians */
   printf("The sine of 30 is:    %f.\n", sin(x));
   printf("The tangent of 30 is: %f.\n", tan(x));

   return 0;
}
```

6. The following is one possible solution (note that I've used the type casting `(double)` in the assignment statement `x=(double)0x19A1;`):

```
/* 09A06.c */
#include <stdio.h>
#include <math.h>
```

B

```
main()
{
   double x;

   x = (double)0x19A1;
   printf("The square root of x is: %2.0f\n", sqrt(x));

   return 0;
}
```

Hour 10, "Controlling Program Flow"

Quiz

1. No.
2. The final result saved in `x` is `2`, after the execution of three cases, `'-'`, `'*'`, and `'/'`.
3. The final result saved in `x` is `2`. This time, only the case of `operator = '-'` is executed due to the `break` statement.
4. The result saved by `x` is `27`.

Exercises

1. The following is one possible solution:

```
/* 10A01.c  Use the if statement */
#include <stdio.h>

main()
{
   int i;

   printf("Integers that can be divided by both 2 and 3\n");
   printf("(within the range of 0 to 100):\n");
   for (i=0; i<=100; i++)
      if (i%6 == 0)
         printf("   %d\n", i);

   return 0;
}
```

2. The following is one possible solution:

```
/* 10A02.c  Use the if statement */
#include <stdio.h>
```

```
main()
{
   int i;

   printf("Integers that can be divided by both 2 and 3\n");
   printf("(within the range of 0 to 100):\n");
   for (i=0; i<=100; i++)
      if (i%2 == 0)
        if (i%3 == 0)
         printf("   %d\n", i);

   return 0;
}
```

3. The following is one possible solution:

```
/* 10A03.c */
#include <stdio.h>

main()
{
   int letter;

   printf("Please enter a letter:\n");
   letter = getchar();
   switch (letter){
      case 'A':
         printf("The numeric value of A is: %d\n", 'A');
         break;
      case 'B':
         printf("The numeric value of B is: %d\n", 'B');
         break;
      case 'C':
         printf("The numeric value of C is: %d\n", 'C');
         break;
      default:
         break;
   }

   return 0;
}
```

4. The following is one possible solution:

```
/* 10A04.c */
#include <stdio.h>

main()
{
   int c;
```

```
   printf("Enter a character:\n(enter q to exit)\n");
   while ((c = getc(stdin)) != 'q') {

     /* no statements inside the while loop */
   }
   printf("\nBye!\n");

   return 0;
}
```

5. The following is one possible solution:

```
/* 10A05.c */
#include <stdio.h>

main()
{
   int i, sum;

   sum = 0;
   for (i=1; i<8; i++){
      if ((i%2 == 0) && (i%3 == 0))
         continue;
      sum += i;
   }
   printf("The sum is: %d\n", sum);
   return 0;
}
```

Hour 11, "Understanding Pointers"

Quiz

1. By using the address-of operator, `&`. That is, the `&ch` expression gives the left value (the address) of the character variable `ch`.
2. The answers are as follows:
   - Dereference operator
   - Multiplication operator
   - Multiplication operator
   - The first and third asterisks are dereference operators; the second asterisk is a multiplication operator.
3. `ptr_int` yields the value of the address `0x1A38`; `*ptr_int` yields the value of `10`.
4. `x` now contains the value of `456`.

Exercises

1. The following is one possible solution:

```
/* 11A01.c */
#include <stdio.h>

main()
{
   int x, y, z;

   x = 512;
   y = 1024;
   z = 2048;

   printf("The left values of x, y, and z are:\n");
   printf("0x%p,  0x%p,  0x%p\n", &x, &y, &z);
   printf("The right values of x, y, and z are:\n");
   printf("%d,  %d,  %d\n", x, y, z);

   return 0;
}
```

2. The following is one possible solution:

```
/* 11A02.c */
#include <stdio.h>

main()
{
   int *ptr_int;
   char *ptr_ch;

   ptr_int = 0; /* null pointer */
   ptr_ch = 0;  /* null pointer */

   printf("The left value of ptr_int is: 0x%p\n",
          ptr_int);
   printf("The right value of ptr_int is: %d\n",
          *ptr_int);
   printf("The left value of ptr_ch is: 0x%p\n",
          ptr_ch);
   printf("The right value of ptr_ch is: %d\n",
          *ptr_ch);

   return 0;
}
```

3. The following is one possible solution:

```
/* 11A03.c */
#include <stdio.h>
```

```
main()
{
   char ch;
   char *ptr_ch;

   ch = 'A';
   printf("The right value of ch is: %c\n",
          ch);
   ptr_ch = &ch;
   *ptr_ch = 'B';  /* decimal 66 */
   /* prove ch has been updated */
   printf("The left value of ch is: 0x%p\n",
          &ch);
   printf("The right value of ptr_ch is: 0x%p\n",
          ptr_ch);
   printf("The right value of ch is: %c\n",
          ch);

   return 0;
}
```

4. The following is one possible solution:

```
/* 11A04.c */
#include <stdio.h>

main()
{
   int x, y;
   int *ptr_x, *ptr_y;

   x = 5;
   y = 6;
   ptr_x = &x;
   ptr_y = &y;

   *ptr_x *= *ptr_y;

   printf("The result is: %d\n",
          *ptr_x);

   return 0;
}
```

Hour 12, "Understanding Arrays"

Quiz

1. It declares an `int` array called `array_int` with four elements. The statement also initializes the array with four integers, `12`, `23`, `9`, and `56`.

2. Because there are only three elements in the `int` array `data`, and the last element is `data[2]`, the third statement is illegal. It may overwrite some valid data in the memory location of `data[3]`.
3. The first array, `array1`, is a two-dimensional array, the second one, `array2`, is one-dimensional, the third one, `array3`, is three-dimensional, and the last one, `array4`, is a two-dimensional array.
4. In a multidimensional array declaration, only the size of the leftmost dimension can be omitted. Therefore, this declaration is wrong. The correct declaration looks like this:

```
char list_ch[][2] = {
     'A', 'a',
     'B', 'b',
     'C', 'c',
     'D', 'd',
     'E', 'e'};
```

Exercises

1. The following is one possible solution:

```
/* 12A01.c */
#include <stdio.h>

main()
{
   int i;
   char array_ch[5] = {'A', 'B', 'C', 'D', 'E'};

   for (i=0; i<5; i++)
     printf("%c ", array_ch[i]);

   return 0;
}
```

2. The following is one possible solution:

```
/* 12A02.c */
#include <stdio.h>

main()
{
   int i;
   char array_ch[5];

   for (i=0; i<5; i++)
      array_ch[i] = 'a' + i;
   for (i=0; i<5; i++)
     printf("%c ", array_ch[i]);

   return 0;
}
```

3. The following is one possible solution:

```
/* 12A03.c */
#include <stdio.h>

main()
{
   int i, size;
   char list_ch[][2] = {
      '1', 'a',
      '2', 'b',
      '3', 'c',
      '4', 'd',
      '5', 'e',
      '6', 'f'};

   /* method I */
   size = &list_ch[5][1] - &list_ch[0][0] + 1;
   size *= sizeof(char);
   printf("Method I: The total bytes are %d.\n", size);

   /* method II */
   size = sizeof(list_ch);
   printf("Method II: The total bytes are %d.\n", size);

   for (i=0; i<6; i++)
     printf("%c  %c\n",
        list_ch[i][0], list_ch[i][1]);

   return 0;
}
```

4. The following is one possible solution:

```
/* 12A04.c */
#include <stdio.h>

main()
{
    char array_ch[11] = {'I', ' ',
                         'l', 'i', 'k', 'e', ' ',
                         'C', '!', '\0'};
    int i;
    /*  array_ch[i] in logical test */
    for (i=0; array_ch[i]; i++)
       printf("%c", array_ch[i]);

    return 0;
}
```

5. The following is one possible solution:

```
/* 12A05.c */
#include <stdio.h>

main()
{
   double list_data[6] = {
         1.12345,
         2.12345,
         3.12345,
         4.12345,
         5.12345};
   int size;

   /* Method I */
   size = sizeof(double) * 6;
   printf("Method I:  The size is %d.\n", size);

   /* Method II */
   size = sizeof(list_data);
   printf("Method II: The size is %d.\n", size);

   return 0;
}
```


Hour 13, "Manipulating Strings"

Quiz

1. The following two statements are legal:
   - `char str2[] = "A character string";`
   - `char str3 = "A";`
2. The following two statements are illegal:
   - `ptr_ch = 'x';`
   - `*ptr_ch = "This is Quiz 2.";`
3. No. The `puts()` function appends a newline character to replace the null character at the end of a character array.
4. The `%s` format specifier is used for reading in a string; the `%f` is for a `float` number.

Exercises

1. The following is one possible solution:

```
/* 13A01.c: Copy a string to another */
#include <stdio.h>
#include <string.h>

main()
{
   int i;
   char str1[] = "This is Exercise 1.";
   char str2[20];

   /* Method I */
   strcpy(str2, str1);

   /* confirm the copying */
   printf("from Method I: %s\n", str2);

   /* Method II */
   for (i=0; str1[i]; i++)
      str2[i] = str1[i];
   str2[i] = '\0';
   /* confirm the copying */
   printf("from Method II: %s\n", str2);

   return 0;
}
```

2. The following is one possible solution:

```
/* 13A02.c: Measure a string */
#include <stdio.h>
#include <string.h>

main()
{
   int i, str_length;
   char str[] = "This is Exercise 2.";

   /* Method I */
   str_length = 0;
   for (i=0; str[i]; i++)
      str_length++;
   printf("The string length is %d.\n", str_length);

   /* Method II */
   printf("The string length is %d.\n",
         strlen(str));

   return 0;
}
```

3. The following is one possible solution:

```
/* 13A03.c: Use gets() and puts() */
#include <stdio.h>

main()
{
   char str[80];
   int i, delt;

   printf("Enter a string less than 80 characters:\n");
   gets( str );
   delt = 'a' - 'A';
   i = 0;
   while (str[i]){
     if ((str[i] >= 'A') && (str[i] <= 'Z'))
        str[i] += delt;  /* convert to lowercase */
     ++i;
   }
   printf("The entered string is (in lowercase):\n");
   puts( str );

   return 0;
}
```

4. The following is one possible solution:

```
/* 13A04.c: Use scanf() */
#include <stdio.h>

main()
{
   int x, y, sum;

   printf("Enter two integers:\n");
   scanf("%d%d", &x, &y);
   sum = x + y;
   printf("The sum is %d\n", sum);

   return 0;
}
```


Hour 14, "Understanding Scope and Storage Classes"

Quiz

1. The `int` variable `x` and `float` variable `y`, declared outside the `myFunction()` function, are global variables. The `int` variables, `i` and `j`, and the `float` variable `y`,

declared inside the function, are local variables. Also, the two `int` variables, `x` and `y`, declared within a block inside `myFunction()`, are local variables with scope limited to the block.

2. For two variables sharing the same name, the compiler can figure out which one to use by checking their scopes. The latest declared variable becomes visible by replacing the variable that has the same name but is declared in the outer block. If, however, two variables sharing the same name are declared in the same block, the compiler will issue an error message.
3. The `int` variable `i` declared outside the `myFunction()` function has the same `static` storage class as the `int` variable `x`. The `float` variable `y` has an `extern` storage class.

   Inside the `myFunction()` function, the two integer variables, `i` and `j`, have the `auto` storage class. The `float` variable `z` has an `extern` storage class, and the `long` variable `s` has a `register` storage class. `index` is an integer variable with a `static` storage class. The content of the character array `str` cannot be changed due to the `const` specifier.
4. No, it's not legal. You cannot change the content of an array specified by the `const` specifier.

Exercises

1. The answers are as follows:
   - `{ int x; }`
   - `{ const char ch; }`
   - `{ static float y; }`
   - `register int z;`
   - `char *ptr_str = 0;`
2. The following is one possible solution:

```
/* 14A02.c */
#include <stdio.h>

int x = 1234;         /* program scope */
float y = 1.234567f;  /* program scope */

void function_1(int x, double y)
{
    printf("From function_1:\n  x=%d, y=%f\n", x, y);
}

main()
```

```
{
    int x = 4321;   /* block scope 1*/

    function_1(x, y);
    printf("Within the main block:\n  x=%d, y=%f\n", x, y);
    /* a nested block */
    {
       float y = 7.654321f;  /* block scope 2 */
       function_1(x, y);
       printf("Within the nested block:\n  x=%d, y=%f\n", x, y);
    }
    return 0;
}
```

3. The following is what I obtained from running the C program given in this exercise:

```
x=0, y=0
x=0, y=1
x=0, y=2
x=0, y=3
x=0, y=4
```

Because `x` has a temporary storage with the block scope, and `y` has a permanent storage, `x` is set to `0` every time the program execution enters the `for` loop, but the value saved in `y` is kept.

4. The following is one possible solution:

```
/* 14A04.c: Use the static specifier */
#include <stdio.h>
/* the add_two function */
int add_two(int x, int y)
{
    static int counter = 1;
    static int sum = 0;

    printf("This is the function call of %d,\n", counter++);
    printf("the previous value of sum is %d,\n", sum);
    sum = x + y;
    return sum;
}
/* the main function */
main()
{
    int i, j;

    for (i=0, j=5; i<5; i++, j--)
       printf("the addition of %d and %d is %d.\n\n",
              i, j, add_two(i, j));
    return 0;
}
```

Hour 15, "Working with Functions"

Quiz

1. The answers are as follows:
   - `int function_1(int x, int y);` is a function declaration with a fixed number of arguments.
   - `void function_2(char *str);` is a function declaration with a fixed number of arguments.
   - `char *asctime(const struct tm *timeptr);` is a function declaration with a fixed number of arguments.
   - `int function_3(void);` is a function declaration with no arguments.
   - `void function_5(void);` is a function declaration with no arguments.
   - `char function_4(char c, ...);` is a function declaration with a variable number of arguments.
2. The second expression is a function definition; that is,

   ```
   int function_2(int x, int y) {return x+y;}
   ```
3. The `int` data type is the default data type returned by a function if a type specifier is omitted.
4. The third one, `char function_3(...);`, is illegal.

Exercises

1. The following is one possible solution:

```
/* 15A01.c: */
 #include <stdio.h>
 #include <time.h>

 void GetDateTime(void);

 main()
 {
    printf("Before the GetDateTime() function is called.\n");
    GetDateTime();
    printf("After the GetDateTime() function is called.\n");
    return 0;
 }
 /* GetDateTime() definition */
 void GetDateTime(void)
 {
    time_t now;
    int i;
```

```
    char *str;

    printf("Within GetDateTime().\n");
    time(&now);
    str = asctime(localtime(&now));
    printf("Current date and time is: ");
    for (i=0; str[i]; i++)
       printf("%c", str[i]);
}
```

2. The following is one possible solution:

```
/* 15A02.c */
#include <stdio.h>

int MultiTwo(int x, int y);

main ()
{
    printf("The result returned by MultiTwo() is: %d\n",
       MultiTwo(32, 10));
    return 0;
}

/* function definition */
int MultiTwo(int x, int y)
{
    return x * y;
}
```

3. The following is one possible solution:

```
/* 15A03.c */
#include <stdio.h>
#include <stdarg.h>

int MultiInt(int x, ...);

main ()
{
    int d1 = 1;
    int d2 = 2;
    int d3 = 3;
    int d4 = 4;

    printf("Given an argument: %d\n", d1);
    printf("The result returned by MultiInt() is: %d\n\n",
       MultiInt(1, d1));
    printf("Given an argument: %d, %d, %d, and %d\n", d1, d2, d3, d4);
    printf("The result returned by MultiInt() is: %d\n\n",
       MultiInt(4, d1, d2, d3, d4));

    return 0;
```

```
}

/* definition of MultiInt() */
int MultiInt(int x, ...)
{
    va_list   arglist;
    int i;
    int result = 1;

    printf("The number of arguments is: %d\n", x);
    va_start (arglist, x);
    for (i=0; i<x; i++)
       result *= va_arg(arglist, int);
    va_end (arglist);
    return result;
}
```

4. The `va_arg()` fetches arguments from left to right on my machine. The following is one possible solution:

```
/* 15A04.c */
#include <stdio.h>
#include <stdarg.h>

double AddDouble(int x, ...);

main ()
{
    double d1 = 1.5;
    double d2 = 2.5;
    double d3 = 3.5;
    double d4 = 4.5;

    printf("Given an argument: %2.1f\n", d1);
    printf("The result returned by AddDouble() is: %2.1f\n\n",
       AddDouble(1, d1));
    printf("Given arguments: %2.1f and %2.1f\n", d1, d2);
    printf("The result returned by AddDouble() is: %2.1f\n\n",
       AddDouble(2, d1, d2));
    printf("Given arguments: %2.1f, %2.1f and %2.1f\n", d1, d2, d3);
    printf("The result returned by AddDouble() is: %2.1f\n\n",
       AddDouble(3, d1, d2, d3));
    printf("Given arguments: %2.1f, %2.1f, %2.1f, and %2.1f\n", d1, d2, d3,
d4);
    printf("The result returned by AddDouble() is: %2.1f\n",
       AddDouble(4, d1, d2, d3, d4));
    return 0;
}

/* definition of AddDouble() */
```

```
double AddDouble(int x, ...)
{
    va_list   arglist;
    int i;
    double argument, result = 0.0;

    printf("The number of arguments is: %d\n", x);
    va_start (arglist, x);
    for (i=0; i<x; i++){
       argument = va_arg(arglist, double);
       printf("Argument passed to this function: %f\n", argument);
       result += argument;
    }

    va_end (arglist);

    return result;
}
```

Hour 16, "Applying Pointers"

Quiz

1. I obtain the following answers from my machine:
   - 4 bytes
   - 4 bytes
   - 4 bytes
   - 12 bytes
   - 12 bytes
   - 12 bytes
2. Because `0x100A - 0x1006` gives `4`, and one `int` takes `2` bytes, `ptr1` and `ptr2` are two integers apart. Therefore, the answer is 2.
3. `0x0230` and `0x0260`.
4. The answers are as follows:
   - `*(ptr + 3)` fetches `'A'`.
   - `ptr - ch` gives `1`.
   - `*(ptr - 1)` fetches `'a'`.
   - `*ptr = 'F'` replaces `'b'` with `'F'`.

Exercises

1. The following is one possible solution:

```
/* 16A01.c */
#include <stdio.h>

void StrPrint(char *str);
main()
{
   char string[] = "I like C!";

   StrPrint(string);

   return 0;
}

void StrPrint(char *str)
{
   printf("%s\n", str);
}
```

2. The following is one possible solution:

```
/* 16A02.c */
#include <stdio.h>

void StrPrint(char *str);
main()
{
   char string[] = "I like C!";
   char *ptr;
   int i;

   ptr = string;
   for (i=0; ptr[i]; i++){
     if (ptr[i] == 'i')
       ptr[i] = 'o';
     if (ptr[i] == 'k')
       ptr[i] = 'v';
   }
   StrPrint(ptr);

   return 0;
}

void StrPrint(char *str)
{
   printf("%s\n", str);
}
```

3. The following is one possible solution:

```
/* 16A03.c */
#include <stdio.h>

void StrPrint(char str[][15], int max);
main()
{
   char str[2][15] = {
        "You know what,",
        "C is powerful." };

   StrPrint(str, 2);

   return 0;
}
void StrPrint(char str[][15], int max)
{
   int i;

   for (i=0; i<max; i++)
     printf("%s\n", str[i]);
}
```

4. The following is one possible solution:

```
* 16A04.c */
#include <stdio.h>

/* function declarations */
void StrPrint1(char **str1, int size);
void StrPrint2(char *str2);

/* main() function */
main()
{
   char *str[7] = {
        "Sunday",
        "Monday",
        "Tuesday",
        "Wednesday",
        "Thursday",
        "Friday",
        "Saturday"};
   int i, size;

   size = 7;
   StrPrint1(str, size);
   for (i=0; i<size; i++)
      StrPrint2(str[i]);
```

```
   return 0;
}

/* function definition */
void StrPrint1(char **str1, int size)
{
   int i;

   for (i=0; i<size; i++)
      printf("%s\n", str1[i]);
}

/* function definition */
void StrPrint2(char *str2)
{
   printf("%s\n", str2);
}
```

Hour 17, "Allocating Memory"

Quiz

1. The answers are as follows:
   - 200 bytes
   - 200 bytes
   - 200 bytes
   - 0 bytes
2. The statement is

   ```
   ptr = realloc(ptr, 150 * sizeof(int));
   ```
3. The final size is 120 bytes, provided the `int` data type is one byte long.
4. The final size is 0. In other words, all allocated memory blocks have been released by the last statement.

Exercises

1. The following is one possible solution:

   ```
   /* 17A01.c */
   #include <stdio.h>
   #include <stdlib.h>

   /* main() function */
   main()
   {
   ```

```
   int *ptr_int;
   int i, sum;
   int max = 0;
   int termination = 0;

   printf("Enter the total number of integers:\n");
      scanf("%d", &max);
   /* call malloc() */
   ptr_int = malloc(max * sizeof(int));
   if (ptr_int == NULL){
     printf("malloc() function failed.\n");
     termination = 1;
   }
   else{
     for (i=0; i<max; i++)
       ptr_int[i] = i + 1;
   }
   sum = 0;
   for (i=0; i<max; i++)
     sum += ptr_int[i];
   printf("The sum is %d.\n", sum);
   free(ptr_int);

   return termination;
}
```

2. The following is one possible solution:

```
/* 17A02.c */
#include <stdio.h>
#include <stdlib.h>

/* main() function */
main()
{
   float *ptr_flt;
   int termination = 0;

   /* call calloc() */
   ptr_flt = calloc(100, sizeof(float));
   if (ptr_flt == NULL){
     printf("calloc() function failed.\n");
     termination = 1;
   }
   else{
     ptr_flt = realloc(ptr_flt, 150 * sizeof(float));
     if (ptr_flt == NULL){
       printf("realloc() function failed.\n");
       termination = 1;
     }
```

```
      else
        free(ptr_flt);
    }
    printf("Done!\n");

    return termination;
}
```

3. The following is one possible solution:

```
/* 17A03.c */
#include <stdio.h>
#include <stdlib.h>

/* main() function */
main()
{
   float *ptr1, *ptr2;
   int i;
   int termination = 1;
   int max = 0;

   printf("Enter the total number:\n");
      scanf("%d", &max);

   ptr1 = malloc(max * sizeof(float));
   ptr2 = calloc(max, sizeof(float));

   if (ptr1 == NULL)
      printf("malloc() failed.\n");
   else if (ptr2 == NULL)
      printf("calloc() failed.\n");
   else{
      for (i=0; i<max; i++)
         printf("ptr1[%d]=%5.2f,  ptr2[%d]=%5.2f\n",
          i, *(ptr1 + i), i, *(ptr2 + i));
      free(ptr1);
      free(ptr2);
      termination = 0;
   }
   printf ("\nBye!\n");

   return termination;
}
```

4. The following is one possible solution:

```
/* 17A04.c: Use the realloc() function */
#include <stdio.h>
#include <stdlib.h>
```

```
#include <string.h>
/* function declaration */
void StrCopy(char *str1, char *str2);
/* main() function */
main()
{
   char *str[4] = {"There's music in the sighing of a reed;",
                   "There's music in the gushing of a rill;",
                   "There's music in all things if men had ears;",
                   "There earth is but an echo of the spheres.\n"
                  };
   char *ptr;
   int i;

   int termination = 0;

   ptr = realloc(NULL, strlen((str[0]) + 1) * sizeof(char));
   if (ptr == NULL){
     printf("realloc() failed.\n");
     termination = 1;
   }
   else{
     StrCopy(str[0], ptr);
     printf("%s\n", ptr);
     for (i=1; i<4; i++){
       ptr = realloc(ptr, (strlen(str[i]) + 1) * sizeof(char));
       if (ptr == NULL){
         printf("realloc() failed.\n");
         termination = 1;
         i = 4;   /* break the for loop */
       }
       else{
         StrCopy(str[i], ptr);
         printf("%s\n", ptr);
       }
     }
   }
   realloc(ptr, 0);
   return termination;
}
/* funciton definition */
void StrCopy(char *str1, char *str2)
{
   int i;

   for (i=0; str1[i]; i++)
      str2[i] = str1[i];
   str2[i] = '\0';
}
```

Hour 18, "Using Special Data Types and Functions"

Quiz

1. The enumerated names `Jan`, `Feb`, `Mar`, `Apr`, `May`, `Jun`, `Jul`, `Aug`, `Sep`, `Oct`, `Nov`, and `Dec` represent the values of `0` to `11`, respectively.
2. The values of `0`, `10`, `11`, and `12` are represented by the enumerated names `name1`, `name2`, `name3`, and `name4`, respectively.
3. The `typedef long int BYTE32;` and `BYTE32 x, y, z;` statements are equivalent to `long int x, y, z;`.

   `typedef char *STRING[16];` and `STRING str1, str2, str3;` are equivalent to `char *str1[16], *str2[16], *str3[16];`
4. No. The `void` keyword in the `main()` function indicates that there is no argument passed to the function.

Exercises

1. The following is one possible solution:

```
/* 18A01.c */
#include <stdio.h>

main(void)
{
   enum tag {name1,
             name2 = 10,
             name3,
             name4 };

   printf("The value represented by name1 is: %d\n",
           name1);
   printf("The value represented by name2 is: %d\n",
           name2);
   printf("The value represented by name3 is: %d\n",
           name3);
   printf("The value represented by name4 is: %d\n",
           name4);

   return 0;
}
```

2. The following is one possible solution:

```
/* 18A02.c */
#include <stdio.h>
```

```
main(void)
{
   typedef char WORD;
   typedef int SHORT;
   typedef long LONG;
   typedef float FLOAT;
   typedef double DFLOAT;

   printf("The size of WORD is: %d-byte\n", sizeof(WORD));
   printf("The size of SHORT is: %d-byte\n", sizeof(SHORT));
   printf("The size of LONG is: %d-byte\n", sizeof(LONG));
   printf("The size of FLOAT is: %d-byte\n", sizeof(FLOAT));
   printf("The size of DFLOAT is: %d-byte\n", sizeof(DFLOAT));

   return 0;
}
```

3. The following is one possible solution:

```
/* 18A03.c */
#include <stdio.h>

enum con{MIN_NUM = 0,
         MAX_NUM = 100};

int fRecur(int n);

main()
{
   int i, sum1, sum2;

   sum1 = sum2 = 0;
   for (i=1; i<=MAX_NUM; i++)
     sum1 += i;
   printf("The value of sum1 is %d.\n", sum1);
   sum2 = fRecur(MIN_NUM);
   printf("The value returned by fRecur() is %d.\n", sum2);

   return 0;
}

int fRecur(int n)
{
   if (n > MAX_NUM)
     return 0;
   return  fRecur(n + 1) + n;
}
```

4. The following is one possible solution:

```
/* 18A04.c: Command-line arguments */
#include <stdio.h>

main (int argc, char *argv[])
```

```
{
   int i;

   if (argc < 2){
     printf("The usage of this program is:\n");
     printf("18A04.EXE argument1 argument2 [...argumentN]\n");
   }
   else {
     printf("The command-line arguments are:\n");
     for (i=1; i<argc; i++)
       printf("%s ", argv[i]);
     printf("\n");
   }

   return 0;
}
```

Hour 19, "Understanding Structures"

Quiz

1. The semicolon (;) should be included at the end of the structure declaration.
2. u, v, and w are three structure variables.
3. You can initialize the array of the `automobile` structure like this:

```
struct automobile {
   int year;
   char model[8]} car[2] = {
       {1997, "Taurus"},
       {1997, "Accord"}};
```

Exercises

1. The following is one possible solution:

```
/* 19A01.c */
#include <stdio.h>

main(void)
{
   struct automobile {
     int year;
     char model[10];
     int engine_power;
     double weight;
     } sedan = {
        1997,
        "New Model",
```

```
        200,
        2345.67};

   printf("year: %d\n", sedan.year);
   printf("model: %s\n", sedan.model);
   printf("engine_power: %d\n", sedan.engine_power);
   printf("weight: %6.2f\n", sedan.weight);

   return 0;
}
```

2. The following is one possible solution:

```
/* 19A02.c */
#include <stdio.h>

struct employee {
   int id;
   char name[32];
};

void Display(struct employee s);

main(void)
{
    /* structure initialization */
    struct employee info = {
       0001,
       "B. Smith"
       };

    printf("Here is a sample:\n");
    Display(info);

    printf("What's your name?\n");
       gets(info.name);
    printf("What's your ID number?\n");
       scanf("%d", &info.id);

    printf("\nHere are what you entered:\n");
    Display(info);

    return 0;
}

/* function definition */
void Display(struct employee s)
{
    printf("Employee Name: %s\n", s.name);
    printf("Employee ID #: %04d\n\n", s.id);
}
```

3. The following is one possible solution:

```
/* 19A03.c Use the -> operator */
#include <stdio.h>

struct computer {
   float cost;
   int year;
   int cpu_speed;
   char cpu_type[16];
};

typedef struct computer SC;

void DataReceive(SC *ptr_s);

main(void)
{
   SC model;

   DataReceive(&model);
   printf("Here are what you entered:\n");
   printf("Year: %d\n", model.year);
   printf("Cost: %6.2f\n", model.cost);
   printf("CPU type: %s\n", model.cpu_type);
   printf("CPU speed: %d MHz\n", model.cpu_speed);

   return 0;
}

void DataReceive(SC *ptr_s)
{
   printf("The type of the CPU inside your computer?\n");
      gets(ptr_s->cpu_type);
   printf("The speed(MHz) of the CPU?\n");
      scanf("%d", &(ptr_s->cpu_speed));
   printf("The year your computer was made?\n");
      scanf("%d", &(ptr_s->year));
   printf("How much you paid for the computer?\n");
      scanf("%f", &(ptr_s->cost));
}
```

4. The following is one possible solution:

```
/* 19L04.c Arrays of structures */
#include <stdio.h>

struct haiku {
   int start_year;
   int end_year;
```

```
   char author[16];
   char str1[32];
   char str2[32];
   char str3[32];
};

typedef struct haiku HK;

void DataDisplay(HK *ptr_s);

main(void)
{
   HK poem[2] = {
     { 1641,
       1716,
       "Sodo",
       "Leading me along",
       "my shadow goes back home",
       "from looking at the moon."
     },
     { 1729,
       1781,
       "Chora",
       "A storm wind blows",
       "out from among the grasses",
       "the full moon grows."
     }
   };

   /* define an array of pointers with HK */
   HK *ptr_poem[2] = {&poem[0], &poem[1]};
   int i;

   for (i=0; i<2; i++)
      DataDisplay(ptr_poem[i]);

   return 0;
}

void DataDisplay(HK *ptr_s)
{
   printf("%s\n", ptr_s->str1);
   printf("%s\n", ptr_s->str2);
   printf("%s\n", ptr_s->str3);
   printf("--- %s\n", ptr_s->author);
   printf("   (%d-%d)\n\n", ptr_s->start_year, ptr_s->end_year);
}
```

Hour 20, "Understanding Unions"

Quiz

1. The first statement is the declaration of a union with the tag name of `_union`. The second statement defines two union variables, `x` and `y`, with the `a_union` data type.
2. The semicolon (`;`) is missed in two places: at the end of the declaration of `char model[8]` and at the end of the declaration of the union.
3. The two union members have the same value, `1997`.

Exercises

1. The following is the modified version. The content of the `name` array is partially overwritten by the value assigned to the `double` variable `price`.

```
/* 20A01.c */
#include <stdio.h>
#include <string.h>

main(void)
{
   union menu {
      char name[23];
      double price;
   } dish;

   printf("The content assigned to the union separately:\n");
   /* access to name */
   strcpy(dish.name, "Sweet and Sour Chicken");
   /* access to price */
   dish.price = 9.95;
   printf("Dish Name: %s\n", dish.name);
   printf("Dish Price: %5.2f\n", dish.price);

   return 0;
}
```

2. The following is one possible solution:

```
/* 20A02.c */
#include <stdio.h>

union employee {
   int start_year;
   int dpt_code;
   int id_number;
};

void DataDisplay(union employee u);
```

```
main(void)
{
    union employee info;

    /* initialize start_year */
    info.start_year = 1997;
    DataDisplay(info);

    /* initialize dpt_code */
    info.dpt_code = 8;
    DataDisplay(info);

    /* initialize id */
    info.id_number = 1234;
    DataDisplay(info);

    return 0;
}

/* function definition */
void DataDisplay(union employee u)
{
    printf("Start Year:  %d\n", u.start_year);
    printf("Dpt. Code:   %d\n", u.dpt_code);
    printf("ID Number:   %d\n", u.id_number);
}
```

OUTPUT The output of the program is

```
Start Year:  1997
Dpt. Code:   1997
ID Number:   1997
Start Year:  8
Dpt. Code:   8
ID Number:   8
Start Year:  1234
Dpt. Code:   1234
ID Number:   1234
```

3. The following is one possible solution:

```
/* 20A03.c */
#include <stdio.h>
#include <string.h>

struct survey {
   char name[20];
   union {
       char state[32];
       char country[32];
   } place;
};
```

```
void DataEnter(struct survey *s);
void DataDisplay(struct survey *s);

main(void)
{
   struct survey citizen;

   DataEnter(&citizen);
   DataDisplay(&citizen);

   return 0;
}

/* definition of DataDisplay() */
void DataDisplay(struct survey *ptr)
{
   printf("\nHere is what you entered: \n");
   printf("Your name is %s.\n", ptr->name);
   printf("You are from %s.\n", ptr->place.state);
   printf("\nThank you!\n");
}

/* definition of DataEnter() */
void DataEnter(struct survey *ptr)
{
   char is_yes[4];

   printf("Please enter your name:\n");
      gets(ptr->name);

   printf("Are you a U. S. citizen? (Yes or No)\n");
      gets(is_yes);
   if ((is_yes[0] == 'Y') ¦¦
       (is_yes[0] == 'y')){
      printf("Enter the name of the state:\n");
      gets(ptr->place.state);
   } else {
      printf("Enter the name of your country:\n");
      gets(ptr->place.country);
   }
}
```

OUTPUT The following is the output of the program on my machine:

```
Please enter your name:
Tony
Are you a U. S. citizen? (Yes or No)
Yes
Enter the name of the state:
Texas
```

```
Here is what you entered:
Your name is Tony.
You are from Texas.

Thank you!
```

4. The following is one possible solution:

```
/* 20A04.c */
#include <stdio.h>
#include <string.h>

struct bit_field {
   int yes: 1;
};

struct survey {
   struct bit_field flag;
   char name[20];
   union {
       char state[32];
       char country[32];
   } place;
};

void DataEnter(struct survey *s);
void DataDisplay(struct survey *s);

main(void)
{
   struct survey citizen;

   DataEnter(&citizen);
   DataDisplay(&citizen);

   return 0;
}

/* function definition */
void DataEnter(struct survey *ptr)
{
   char is_yes[4];

   printf("Please enter your name:\n");
      gets(ptr->name);

   printf("Are you a U.S. citizen? (Yes or No)\n");
      gets(is_yes);
   if ((is_yes[0] == 'Y') ¦¦
       (is_yes[0] == 'y')){
      printf("Enter the name of the state:\n");
      gets(ptr->place.state);
```

```
      ptr->flag.yes = 1;
   } else {
      printf("Enter the name of your country:\n");
      gets(ptr->place.country);
      ptr->flag.yes = 0;
   }
}
/* function definition */
void DataDisplay(struct survey *ptr)
{
   printf("\nHere is what you've entered:\n");
   printf("Name: %s\n", ptr->name);
   if (ptr->flag.yes)
      printf("The state is: %s\n",
          ptr->place.state);
   else
      printf("Your country is: %s\n",
          ptr->place.country);
   printf("\nThanks and Bye!\n");
}
```

Hour 21, "Reading and Writing with Files"

Quiz

1. The first expression tries to open an existing binary file called `test.bin` for reading and writing. The second expression tries to open an existing text file called `test.txt` for appending. The last expression tries to create a text file, called `test.ini`, for reading and writing.
2. The `fopen()` function returns a null pointer when an error occurs during the procedure of opening a file. It's not legal to do any reading or writing with a null file pointer. Therefore, the code is wrong because it calls `fgetc()` when `fopen()` returns a null pointer.
3. The mode is set to read only, but the code tries to write a character to the opened file by calling the `fputc()` function.
4. The code still reads a text file by using the file pointer `fptr1`, even though the file pointer `fptr1` has been closed.

Exercises

1. The following is one possible solution:

```
/* 21A01.c */
#include <stdio.h>
```

```
enum {SUCCESS, FAIL};

int CharRead(FILE *fin);

main(void)
{
   FILE *fptr;
   char filename[]= "haiku.txt";
   int reval = SUCCESS;

   if ((fptr = fopen(filename, "r")) == NULL){
      printf("Cannot open %s.\n", filename);
      reval = FAIL;
   } else {
      printf("\nThe total character number is %d.\n",
          CharRead(fptr));
      fclose(fptr);
   }

   return reval;
}

/* definition of CharRead() */
int CharRead(FILE *fin)
{
   int c, num;

   num = 0;
   while ((c=fgetc(fin)) != EOF){
      putchar(c);
      ++num;
   }

   return num;
}
```


2. The following is one possible solution:

```
/* 21A02.c */
#include <stdio.h>
#include <string.h>

enum {SUCCESS, FAIL, MAX_LEN = 80};

void LineWrite(FILE *fout, char *str);

main(void)
{
   FILE *fptr;
   char str[MAX_LEN+1];
   char filename[32];
   int reval = SUCCESS;
```

```
   printf("Please enter the file name:\n");
   gets(filename);

   printf("Enter a string:\n");
   gets(str);

   if ((fptr = fopen(filename, "w")) == NULL){
      printf("Cannot open %s for writing.\n", filename);
      reval = FAIL;
   } else {
      LineWrite(fptr, str);
      fclose(fptr);
   }

   return reval;
}

/* definition of LineWrite() */
void LineWrite(FILE *fout, char *str)
{
   fputs(str, fout);
   printf("Done!\n");
}
```

3. The following is one possible solution:

```
/* 21A03.c */
#include <stdio.h>

enum {SUCCESS, FAIL};

void CharWrite(FILE *fout, char *str);

main(void)
{
   FILE *fptr;
   char filename[]= "test_21.txt";
   char str[]= "Disk file I/O is fun.";
   int reval = SUCCESS;

   if ((fptr = fopen(filename, "w")) == NULL){
      printf("Cannot open %s.\n", filename);
      reval = FAIL;
   } else {
      CharWrite(fptr, str);
      fclose(fptr);
   }

   return reval;
}
```

```
/* function definition */
void CharWrite(FILE *fout, char *str)
{
   int i, c;

   i = 0;
   while ((c=str[i]) != '\0'){
      putchar(c);
      fputc(c, fout);
      i++;
   }
}
```

4. The following is one possible solution:

```
/* 21A04.c */
#include <stdio.h>
#include <string.h>

enum {SUCCESS, FAIL};

void BlkWrite(FILE *fout, char *str);

main(void)
{
   FILE *fptr;
   char filename[]= "test_21.txt";
   char str[]= "Disk file I/O is tricky.";
   int reval = SUCCESS;

   if ((fptr = fopen(filename, "w")) == NULL){
      printf("Cannot open %s.\n", filename);
      reval = FAIL;
   } else {
      BlkWrite(fptr, str);
      fclose(fptr);
   }

   return reval;
}

/* function definition */
void BlkWrite(FILE *fout, char *str)
{
   int num;

   num = strlen(str);
   fwrite(str, sizeof(char), num, fout);
   printf("%s\n", str);
}
```

Hour 22, "Using Special File Functions"

Quiz

1. Yes. The two statements are equivalent.
2. No. The two statements are not equivalent, unless the current file position indicator is indeed at the beginning of the file.
3. The `scanf()` function reads from the `test.txt` file, instead of the default input stream, because the `freopen()` function has redirected the input stream and associated it with the `test.txt` file.
4. The four `double` data items together are going to take 32 bytes in the binary file, if the size of the `double` data type is eight bytes long.

Exercises

1. The following is one possible solution:

```
/* 22A01.c */
#include <stdio.h>

enum {SUCCESS, FAIL, MAX_LEN = 80};

void PtrSeek(FILE *fptr);
long PtrTell(FILE *fptr);
void DataRead(FILE *fptr);
int ErrorMsg(char *str);

main(void)
{
    FILE *fptr;
    char filename[]= "LaoTzu.txt";
    int reval = SUCCESS;

    if ((fptr = fopen(filename, "r")) == NULL){
       reval = ErrorMsg(filename);
    } else {
       PtrSeek(fptr);
       fclose(fptr);
    }

    return reval;
}

/* function definition */
void PtrSeek(FILE *fptr)
{
    long offset1, offset2, offset3;
```

```
    offset1 = PtrTell(fptr);
    DataRead(fptr);
    offset2 = PtrTell(fptr);
    DataRead(fptr);
    offset3 = PtrTell(fptr);
    DataRead(fptr);
    printf("\nRe-read the paragraph:\n");

    /* re-read the third sentence */
    fseek(fptr, offset3, SEEK_SET);
    DataRead(fptr);

    /* re-read the second sentence */
    fseek(fptr, offset2, SEEK_SET);
    DataRead(fptr);

    /* re-read the first sentence */
    fseek(fptr, offset1, SEEK_SET);
    DataRead(fptr);
}

/* function definition */
long PtrTell(FILE *fptr)
{
    long reval;

    reval = ftell(fptr);
    printf("The fptr is at %ld\n", reval);

    return reval;
}

/* function definition */
void DataRead(FILE *fptr)
{
    char buff[MAX_LEN];

    fgets(buff, MAX_LEN, fptr);
    printf("%s", buff);
}

/* function definition */
int ErrorMsg(char *str)
{
    printf("Cannot open %s.\n", str);

    return FAIL;
}
```

2. The following is one possible solution:

```
/* 22A02.c */
#include <stdio.h>

enum {SUCCESS, FAIL, MAX_LEN = 80};

void PtrSeek(FILE *fptr);
long PtrTell(FILE *fptr);
void DataRead(FILE *fptr);
int ErrorMsg(char *str);

main(void)
{
    FILE *fptr;
    char filename[]= "LaoTzu.txt";
    int reval = SUCCESS;

    if ((fptr = fopen(filename, "r")) == NULL){
       reval = ErrorMsg(filename);
    } else {
       PtrSeek(fptr);
       fclose(fptr);
    }

    return reval;
}

/* function definition */
void PtrSeek(FILE *fptr)
{
    long offset1, offset2, offset3;

    offset1 = PtrTell(fptr);
    DataRead(fptr);
    offset2 = PtrTell(fptr);
    DataRead(fptr);
    offset3 = PtrTell(fptr);
    DataRead(fptr);
    printf("\nRe-read the paragraph:\n");
    /* re-read the third sentence */
    fseek(fptr, offset3, SEEK_SET);
    DataRead(fptr);
    /* re-read the second sentence */
    fseek(fptr, offset2, SEEK_SET);
    DataRead(fptr);
    /* re-read the first sentence */
    rewind(fptr);  /* rewind the file position indicator */
    DataRead(fptr);
}
```

```
/* function definition */
long PtrTell(FILE *fptr)
{
    long reval;

    reval = ftell(fptr);
    printf("The fptr is at %ld\n", reval);

    return reval;
}

/* function definition */
void DataRead(FILE *fptr)
{
    char buff[MAX_LEN];

    fgets(buff, MAX_LEN, fptr);
    printf("%s", buff);
}

/* function definition */
int ErrorMsg(char *str)
{
    printf("Cannot open %s.\n", str);
    return FAIL;
}
```


3. On my machine, the `data.bin` binary file is 10 bytes. The following is one possible solution:

```
/* 22A03.c */
#include <stdio.h>

enum {SUCCESS, FAIL};

void DataWrite(FILE *fout);
void DataRead(FILE *fin);
int ErrorMsg(char *str);

main(void)
{
   FILE *fptr;
   char filename[]= "data.bin";
   int reval = SUCCESS;

   if ((fptr = fopen(filename, "wb+")) == NULL){
      reval = ErrorMsg(filename);
   } else {
      DataWrite(fptr);
      rewind(fptr);
      DataRead(fptr);
```

```
      fclose(fptr);
   }

   return reval;
}

/* function definition */
void DataWrite(FILE *fout)
{
   double dnum;
   int inum;

   dnum = 123.45;
   inum = 10000;

   printf("%5.2f\n", dnum);
   fwrite(&dnum, sizeof(double), 1, fout);
   printf("%d\n", inum);
   fwrite(&inum, sizeof(int), 1, fout);
}

/* function definition */
void DataRead(FILE *fin)
{
   double x;
   int y;

   printf("\nRead back from the binary file:\n");
   fread(&x, sizeof(double), (size_t)1, fin);
   printf("%5.2f\n", x);
   fread(&y, sizeof(int), (size_t)1, fin);
   printf("%d\n", y);
}

/* function definition */
int ErrorMsg(char *str)
{
   printf("Cannot open %s.\n", str);
   return FAIL;
}
```

4. The following is one possible solution:

```
/* 22A04.c */
#include <stdio.h>

enum {SUCCESS, FAIL,
      MAX_NUM = 3,
      STR_LEN = 23};

void DataRead(FILE *fin);
int ErrorMsg(char *str);
```

```
main(void)
{
   FILE *fptr;
   char filename[]= "strnum.mix";
   int reval = SUCCESS;

   if ((fptr = freopen(filename, "r", stdin)) == NULL){
      reval = ErrorMsg(filename);
   } else {
      DataRead(fptr);
      fclose(fptr);
   }

   return reval;
}

/* function definition */
void DataRead(FILE *fin)
{
   int i;
   int miles;
   char cities[STR_LEN];

   printf("The data read:\n");
   for (i=0; i<MAX_NUM; i++){
      scanf("%s%d", cities, &miles);
      printf("%-23s  %d\n", cities, miles);
   }
}

/* function definition */
int ErrorMsg(char *str)
{
   printf("Cannot open %s.\n", str);
   return FAIL;
}
```


Hour 23, "Compiling Programs: The C Preprocessor"

Quiz

1. The semicolon (;) should not be included at the end of the macro definition because a macro definition ends with a newline, not a semicolon.
2. The value of `82` is assigned to `result` due to the assignment expression `result = 1 + 9 * 9`.

3. The message of `Under #else.` is printed out.
4. The message of `Under #ifdef.` is printed out.

Exercises

1. The following is one possible solution:

```
/* 23A01.c */
#include <stdio.h>
/* main() function */
main()
{
   #define human     100
   #define animal    50
   #define computer  51
   #define SUN       0
   #define MON       1
   #define TUE       2
   #define WED       3
   #define THU       4
   #define FRI       5
   #define SAT       6


   printf("human: %d,  animal: %d,  computer: %d\n",
      human, animal, computer);
   printf("SUN: %d\n", SUN);
   printf("MON: %d\n", MON);
   printf("TUE: %d\n", TUE);
   printf("WED: %d\n", WED);
   printf("THU: %d\n", THU);
   printf("FRI: %d\n", FRI);
   printf("SAT: %d\n", SAT);

   return 0;
}
```

2. The following is one possible solution:

```
/* 23A02.c */
#include <stdio.h>

#define MULTIPLY(val1, val2)  ((val1) * (val2))
#define NO_ERROR  0

main(void)
{
   int result;

   result = MULTIPLY(2, 3);
```

```
   printf("MULTIPLY(2, 3) produces value of %d.\n", result);

   return NO_ERROR;
}
```

3. The following is one possible solution:

```
/* 23A03.c */
#include <stdio.h>

#define UPPER_CASE  0
#define NO_ERROR    0

main(void)
{
   #if UPPER_CASE
     printf("THIS LINE IS PRINTED OUT,\n");
     printf("BECAUSE UPPER_CASE IS DEFINED.\n");
   #elif LOWER_CASE
     printf("This line is printed out,\n");
     printf("because LOWER_CASE is defined.\n");
   #else
     printf("This line is printed out,\n");
     printf("because neither UPPER_CASE nor LOWER_CASE is defined.\n");
   #endif

   return NO_ERROR;
}
```

4. The following is one possible solution:

```
/* 23A04.c:   */
#include <stdio.h>

#define C_LANG    'C'
#define B_LANG    'B'
#define NO_ERROR  0

main(void)
{
   #if C_LANG == 'C'
      #if B_LANG == 'B'
        #undef C_LANG
        #define C_LANG "I know C language.\n"
        #undef B_LANG
        #define B_LANG "Also, I know BASIC.\n"
        printf("%s%s", C_LANG, B_LANG);

     #else
        #undef C_LANG
        #define C_LANG "I only know C language.\n"
        printf("%s", C_LANG);
```

```
    #endif
  #elif B_LANG == 'B'
    #undef B_LANG
    #define B_LANG "I only know BASIC.\n"
    printf("%s", B_LANG);
  #else
    printf("I don't know C or BASIC.\n");
  #endif

  return NO_ERROR;
}
```

INDEX

Symbols

A

C

D

E

F

G

H

I

J-K

L

M

N

O

P

Q-R

S

T

U

V

W-Z