Chapter 14

14.1. The function main() is the place in a C program where execution begins. A program without a function main() has no starting point.

14.3. a. The function declaration informs the compiler about the return type, input parameters, and function name for a given function. This information is required so that the compiler can generate code for a function call to this function if it appears in the source code prior to the function definition.

b. A function prototype is the same as function definition.

c. A function definition contains the source code for a function.

d. An argument is an input value for a callee function.

e. A parameter is a value provided by the caller function for at callee function.

14.5. The output of the program is "2 2". The variable 'z' in the function MyFunc() is declared within the local scope of the function. The value of 'z' from main() is passed to MyFunc(), but all operations on 'z' in MyProc affect only the local copy and not the one in main().

14.7.

Activation Record
int a
dynamic link
return address
return value
int x

Description	Writer
local variable	Bump
address of data	Bump
address of instruction	Bump
other	Bump
argument	---

14.9. The parameters are placed onto the stack before the JSR is called. This is necessary because once the callee is called, the original data values in the caller are unavailable. The caller's data is out of scope once the JSR is called.

14.11.a. a = 3 b = ??? (b is an unknown value)

b. The local variable z is uninitialized and therefore can have any arbitrary value. However, since the position of the activation record for function Unit() on the run-time stack corresponds to the position of the function Init(), which was called previously, the value of local variable z will correspond to the value 2 (in other words, variable z reuses the location allocated to variable y in function Init).

14.13.

```
#include <stdio>

void PrintBase4(int x);

int main()
{
      int a, b;

      printf("First Number: ");
      scanf("%d", &a);
      printf("Second Number: ");
      scanf("%d", &b);

      PrintBase4(a);
      PrintBase4(b);
      PrintBase4(a + b);
}

void PrintBase4(int x)
{
      int i;
      int digits = -1;
      int temp = x;

      /* find out how many digits in the number */
      if (x > 0)
            while ( temp )
            {
                  temp = temp / 4;
                  digits++;

            }
      else
            digits = 0;

      /* print out digits from highest down to lowest */
      for ( digits = digits; digits >= 0; digits--)
      {
            temp = 1;
            for ( i = 0; i < digits; i++)
                  temp = temp * 4;
            temp = (x / temp) % 4;

            printf("%d",temp);
      }
      printf("\n");

}
```

14.15.

Run-time Stack
16 (int x1)
dynamic link for main
x3103 (return addr to main)
0 (return value from f)
4 (third arg to f)
5 (second arg to f)
6 (first arg to f)
6 (int c)
5 (int b)
4 (int a)

14.17.

```
int Multiplex(int input0, int input1, int input2, int input3, int select)
            {
                  switch (select)
                  {
                        case 0:
                              return input0;
                              break;
                        case 1:
                              return input1;
                              break;
                        case 2:
                              return input2;
                              break;
                        case 3:
                              return input3;
                              break;
                        default:
                              return 0;
                              break;
                  }
            }

            int Alu(int input0, int input1, int select)
            {
                  switch (select)
                  {
                  case ALU_ADD:
                        return (input0 + input1);
                        break;
                  case ALU_AND:
                        return (input0 & input1);
                        break;
                  case ALU_NOT:
                        return (~input0);
                        break;
                  default:
                        return 0;
                        break;
                  }
            }
```

14.19.

```
A: The variable z equals 3
B: The variable z equals 5
C: The variable z equals 7
D: The variable z equals 4
```

Questions in the text denoted by the question mark icon:

Page 385: As discussed later in the chapter, an activation record for a function is allocated on the run-time stack.

Page 397: The following is a straightforward technique to calculate Pythagorean Triples more efficiently than the code in Figure 14.11. Why is this technique more efficient? Notice the loop bounds on the nested for loops. There techniques for making the code even more efficient than this using an algebraic reduction on the Pythagorean relationship.

```
#include <stdio.h>

int Squared(int x);

int main()
{
  int sideA;
  int sideB;
  int sideC;
  int maxC;

  printf("Enter the maximum length of hypotenuse: ");
  scanf("%d", &maxC);

  for (sideC = 1; sideC <= maxC; sideC++) {
    for (sideB = 1; sideB <= sideC; sideB++) {
      for (sideA = 1; sideA <= sideB; sideA++) {
        if (Squared(sideC) == Squared(sideA) + Squared(sideB))
          printf("%d %d %d\n", sideA, sideB, sideC);
      }
    }
  }
}

/* Calculate the square of a number */
int Squared(int x)
{
  return x * x;
}
```