CHAPTER 9

Programming Languages

(Solutions to Practice Set)

Review Questions

1. A machine language uses only 0s and 1s for instructions and addresses. An assembly language uses symbols to represent instructions and addresses.
2. A high-level languages is more close to a natural language such as English. An assembly language is the machine language in which the instructions and addresses in binary are replaced by symbols.
3. The machine language is the only language understood by the computer hardware.
4. Compilation translates the whole source program into the object module before executing the program. Interpretation refers to process of translating each line of the source program into the corresponding line of the object program and executing the translated line.
5. The four steps are lexical analysis, syntax analysis, semantic analysis, and code generation.
6. The four common paradigms are procedural, object-oriented, functional, and declarative.
7. In the procedural paradigm, a program is an active agent that manipulates passive objects (data). In an object-oriented paradigm, data are designed as active objects. The action to be performed on these objects are included in the object.
8. In the object-oriented paradigm, a class is the definition of a set of objects. In these languages, a method is an action that can be performed by an instance of the class (an object).
9. In the functional paradigm a program is designed like a mathematical function. It allows the programmer to combine predefined primitive functions to create new functions.
10. In the declarative paradigm a program uses the principle of logical reasoning to answer queries.

Multiple-Choice Questions

11. a	**12.** c	**13.** b	**14.** a	**15.** a	**16.** d
17. b	**18.** d	**19.** c	**20.** d	**21.** b	**22.** c

Exercises

23.

```
int count;
int index;
int level;
```

24.

```
float tax = 0.08;
float price = 12.32;
float sum = 0.00
```

25.

```
const char name =’A’;
const int count = 1;
const float height = 1.82;
```

26. If a constant is not initialized, it is useless because it cannot take a value later.

27. The statement is executed twice (once when A = 5 and the second time when A = 7). When A becomes 9, the loop is terminated.

28. The statement is executed infinite number of times (A = 5, 3, 1, −1, −3,...).

29. The statement is executed eight times (*i* = 5, 7, 9, 11, 13, 15, 17, 19). Note that in each iteration the value of *i* is incremented twice: the first time inside the header (*i*++), the second time in the body of the loop (*i* = *i* + 1).

30. The statement is executed five times (A = 5, 6, 7, 8, 9)

31.

```
A = 5;
do
{
        statement;
        A = A – 2;
} while (A < 8);
```

32.

```
i = 5;
do
{
        statement;
        i = i + 2;
} while (i < 20);
```

33.

```
i = 5;
while (i < 20)
{
        statement;
        i = i + 2;
}
```

34.

```
for (int A = 5; A < 10; A++)
{
        statement;
}
```

35.

```
for (int A = 5; A < 8; A = A – 2)
{
        statement;
}
```

36.

```
A = 0;
while (A > 0)
{
        statement;
        A = A – 1;
}
```

37. This is not possible because in a *do-while* loop, the body of the loop is executed at least once.

38. The following shows one possible solution.

```
for (int i = 0; i < 0; i– –)
{
        statement;
}
```

39. The following shows one possible solution.

```
while (true)
{
        statement;
}
```

40. The following shows one possible solution.

```
do
{
        statement;
} while (true);
```

41. The following shows one possible solution.

```
for (; true; )
{
        statement;
}
```

42. The literal values are 12, 4, and 5.

43. *Hello* is the variable, "Hello" is the literal.

44.

```
switch (A)
        {
                case 4:     statement 1;
                            break;
                case 6:     statement 2;
                            break;
                case 8:     statement 3;
                            break;
        }
```

Note that we don’t need the last *break* statement, but is normally it is included to make each case the same.

45. A and B should be passed by value, S and P by reference.

46.

- **a.** A and B should be passed by value, S by reference.
- **b.** Alternatively, we can pass A and B by value and let the function **smaller** return the smaller value. The following shows how the **smaller** function is called:

```
sm ← smaller (A, B);
```

47.

- **a.** It should be by reference if we can allow the subprogram change the value of A in the main program. The following shows the statement:

```
cube (A);
```

- **b.** Alternatively, we can pass A by value and let the function **cube** return the cube of A. In this case, the original value of A remains untouched in the main program. The following shows the statement:

```
result ← cube (A);
```

48. It should be passed by reference because the flow is from the subprogram to the main program. The subprogram, after getting the value from the keyboard, needs to pass it to the main program. Pass by value cannot be used in this case.

49. It can be passed either by value or by reference, but it is normally passed by value to keep the value of the variable in the main untouched.