Introduction to Programming Languages

Programming in C, C++, Scheme, Prolog, C#, and SOA

Sixth Edition

Answer Keys to Multiple Choice Questions

Yinong Chen

Arizona State University

Kendall Hunt
publishing company

1.7 Homework and programming exercises

1. Multiple Choice. Choose only one answer for each question. Choose the best answer if more than one answer is acceptable.

1.1

❑ imperative ❑ object-oriented ❑ functional ❑ logic

1.2

❑ imperative ❑ functional ❑ object-oriented ❑ logic

1.3

❑ imperative ❑ object-oriented ❑ functional ❑ service-oriented

1.4

❑ stateless ❑ state encapsulation ❑ platform-independent ❑ side-effect free

1.5

❑ Web is the computing platform ❑ Web supports graphic display
❑ Web supports semantic analysis ❑ Web is accessed over HTTP protocol

1.6

❑ efficiency ❑ orthogonality ❑ reliability ❑ readability

1.7

❑ readability ❑ writability ❑ efficiency ❑ None

1.8

❑ reduce the types of control structures. ❑ increase the types of control structures.
❑ make programs execute faster. ❑ use BNF to define the syntactic structure.

1.9

❑ typing coercion casting paradigm

1.10

```
int i = 1; char c = 'a';   // declaration and initialization
c = c + i;                 // execution of an assignment statement
```

❑ Ada ❑ C ❑ Java ❑ All of them

1.11

❑ lexical ❑ syntactic ❑ contextual ❑ semantic

1.12

```
<char>        ::= a | b | c | ... | x | y | z
<identifier>  ::= <char> | <char> <identifer>
```

❑ 0 ❑ 1 ❑ 26 ❑ more than 26

1.13

- ❑ Each variable in a program has a single type associated with it.
- ❑ Variable type can be unknown at compilation time.
- ❑ Type errors are always reported.
- ❑ Coercion is automatically allowed.

1.14

❑ if-else ❑ switch ❑ for ❑ while

1.15

- ❑ how to form lexical units from characters.
- ❑ how to put lexical units together to form statements.
- ❑ the static semantics that will be checked by the compiler.
- ❑ the dynamic semantics of programs under execution.

1.16

❑ lexical error ❑ syntactic error ❑ contextual error ❑ semantic error

1.17

- ❑ the source program is small.
- ❑ the source program is written in an assembly language.
- ❑ the difference between source and destination is small.
- ❑ multi-module programming is used.

1.18

- ❑ There is no difference between them.
- ❑ The *inline* functions are for Java only. There are no inline functions in C++.
- ❑ *Inlining* is a suggestion to the compiler, while a *macro* definition will be enforced.
- ❑ A *macro* definition is a suggestion to the compiler, while *inlining* will be enforced.

1.19

❑ Editing ❑ Preprocessing ❑ Compilation ❑ Execution

1.20

- ❑ a longer machine code but with shorter execution time.
- ❑ shorter machine code but with longer execution time.
- ❑ the same length of machine code, with shorter execution time.
- ❑ the same length of machine code, with longer execution time.

2.12 Homework, programming exercises, and projects

1. Multiple Choice. Choose only one answer for each question. Choose the best answer if more than one answer is acceptable.

1.1

- ❑ provides a level of abstraction.
- ❑ is never necessary.
- ❑ is not required if `<iostream>` is included.
- ❑ is useless.

1.2

- ❑ C is not designed to handle logic operations.
- ❑ C uses strong type checking.
- ❑ Boolean values can be represented as integers.
- ❑ C++ has already defined a Boolean type.

1.3

- ❑ one function is defined within the other function.
- ❑ they call each other.
- ❑ each function calls itself.
- ❑ they are independent of each other .

1.4

❑ 1 ❑ 5 ❑ 6 ❑ 7 ❑ 40

1.5

❑ `str2 = str1;` ❑ `str2 = 0;` ❑ `str1 = str1+1;` ❑ `*str2 = "Hi";`

1.6

- ❑ `char s[5];`
- ❑ `char s[3] = "hello";`
- ❑ `char s[];`
- ❑ `char s[] = {'s', 't', 'r'};`

1.7

❑ `j++;` ❑ `k++;` ❑ `(*k)++;` ❑ `(**k)++;`

1.8

❑ `i++;` ❑ `(&i)++;` ❑ `(*j)++;` ❑ `(**k)++;`

1.9

❑ `1` ❑ `2` ❑ `3` ❑ `more than 3`

1.10

❑ bool ❑ double ❑ int ❑ string

1.11

❑ array file ❑ binary file ❑ text file ❑ structure file

1.12

- ❑ output leaves a character in the file buffer.
- ❑ output fails to complete its operation.

❑ input leaves a character in the file buffer. ❑ input fails to complete its operation.

1.13

❑ 4 bytes ❑ 66 bytes ❑ 68 bytes ❑ 72 bytes

what is the size of x?

❑ 4 bytes ❑ 66 bytes ❑ 68 bytes ❑ 72 bytes

1.14

❑ call-by-value ❑ call-by-alias ❑ call-by-address ❑ None of them

1.15

❑ call-by-value ❑ call-by-alias ❑ call-by-address ❑ None of them

1.16

❑ call-by-value ❑ call-by-alias ❑ call-by-address ❑ None of them

1.17

❑ head-recursion ❑ middle-recursion ❑ tail-recursion ❑ mutual recursion

1.18

❑ (s, t) ❑ A(s, t) ❑ A(n) ❑ n

1.19

❑ s = 0 and t+1 ❑ s = 0 and t = 1 ❑ s > 0 and t = 0 ❑ s > 0 and t > 0

1.20

❑ A(s, t) ❑ A(s-1, 1) ❑ A(s, t-1) ❑ A(s-1, A(s, t-1))

1.21

❑ Preorder ❑ postorder ❑ inorder ❑ in any order

1.22

❑ binary search tree ❑ array ❑ doubly linked list ❑ linked list

3.11 Homework, programming exercises, and projects

1. Multiple Choice. Choose only one answer for each question. Choose the best answer if more than one answer is acceptable.

1.1

- ❑ a pointer to a class `A` object can point to a class `B` object.
- ❑ a pointer to a class `B` object can point to a class `A` object.
- ❑ a pointer to a class `A` object can point to a class `B` object, and vice versa.
- ❑ a pointer to a class `A` object can NOT point to a class `B` object, and vice versa.

1.2

- ❑ is an abstract interface that has no implementation.
- ❑ is an extendable function that allows a programmer to add formal parameters to the function.
- ❑ implies early binding between the function name and the code.
- ❑ implies late binding between the function name and the code.

1.3

- ❑ any members of the derived class object.
- ❑ the inherited members of the derived class object.
- ❑ the additional members of the derived class object.
- ❑ public members in the derived class object.

1.4

- ❑ heap object created in the constructor.
- ❑ heap object created in main() function.
- ❑ stack object created in constructor.
- ❑ static object created in main() function.

1.5

- ❑ heap object created in the constructor
- ❑ heap object created in main() function
- ❑ stack object created in constructor
- ❑ static object created in main() function

1.6

- ❑ Only in `A`.
- ❑ Only in `B`.
- ❑ In `A` and `B`.
- ❑ None of them.

1.7

- ❑ Use public.
- ❑ Use protected.
- ❑ Use private.
- ❑ Use friend.

1.8

- ❑ `p = q;`
- ❑ `q = p;`
- ❑ Both of them.
- ❑ None of them.

1.9

- ❑ cannot be defined in class `A`.
- ❑ can be re-defined in a derived class.
- ❑ cannot be re-defined in a derived class.
- ❑ none of the above.

1.10

- ❑ any members of the derived class object.
- ❑ the inherited members of the derived class object.
- ❑ the new (extended) members of the derived class object.
- ❑ public members in the derived class object.

1.11

- ❑ derive one class from the other (use inheritance).
- ❑ contain one class in the other.
- ❑ define them totally independent of each other.
- ❑ none of the above

1.12

- ❑ number of data fields in classes.
- ❑ number of member functions in classes.
- ❑ number of public members in classes.
- ❑ inheritance relationship among classes.

1.13

❑ zero or more. ❑ one only. ❑ one or more. ❑ at most two.

1.14

- ❑ exit from a try-block and pass a value to a catch-block
- ❑ exit from a catch-block and pass a value to the try-block
- ❑ same as try
- ❑ none of the above

1.15

❑ `manager1` ❑ `manager2` ❑ both of them ❑ none of them

1.16

❑ `m.id` ❑ `m.empl.id` ❑ `m.empl->id`

1.17

❑ `n.id` ❑ `n.empl.id` ❑ `n.empl->id`

1.18

❑ static_cast ❑ const_cast ❑ dynamic_cast ❑ reinterpret_cast

1.19

❑ static_cast ❑ const_cast ❑ dynamic_cast ❑ reinterpret_cast

1.20

❑ virtual function ❑ function overloading ❑ operator overloading ❑ virtual operator

1.21

❑ Primitive type ❑ String type ❑ Object types ❑ All of them.

1.22

❑ A type that can take different types of values at the same time.
❑ A type that contains all other types.
❑ A type that takes memory from heap.
❑ The type can be determined at run time.

1.23

❑ Yes ❑ No

1.24

❑ To improve the performance of multithreading.
❑ To make sure that all the threads complete their tasks.
❑ To instantiate a concrete type to a generic type.
❑ To avoid a deadlock.

1.25

❑ defining generic type. ❑ defining generic class.
❑ defining parallel computing process. ❑ saving memory usage.

2.1

❑ enqueue(int v) ❑ dequeue(void) ❑ Queue(int n) ❑ ~Queue(void)

2.2

❑ front ❑ rear ❑ queue_size ❑ *buffer

2.3

❑ static memory ❑ stack ❑ heap ❑ queue

2.4

❑ in the constructor ❑ in the destructor ❑ in the `enqueue` function
❑ in the `dequeue` function ❑ implicitly in `delete myQueue`

2.5

❑ static memory ❑ stack ❑ heap ❑ queue

2.6

❑ static memory ❑ stack ❑ heap ❑ queue

4.11 Homework, programming exercises, and projects

1. Multiple Choice. Choose only one answer for each question. Choose the best answer if more than one answer is acceptable.

1.1

❑ `#t` ❑ `#f` ❑ `A` ❑ error message

1.2

❑ #f ❑ (3 5 2 9) ❑ (4 2 1) ❑ (2 1)

1.3

❑ `(6 8 10)` ❑ `(6)` ❑ `6` ❑ error message

1.4

❑ (NULL? L)
❑ (< (length L) 1)
❑ (= (length L) 0)
❑ (= L 0)

1.5

❑ (+ z 3)
❑ (define foo (lambda (z) (+ z 3)))
❑ none of them
❑ ((lambda (z) (+ z 3)) 4)
❑ (define bar 25)

1.6

❑ all parameters of a function first.
❑ a parameter of a function only if it is necessary.
❑ no parameters at all.
❑ outermost first.

1.7

❑ all parameters of a function first.
❑ a parameter of a function only if it is necessary.
❑ innermost first.
❑ no parameters at all.

1.8

❑ may produce different results.
❑ never produce different results.
❑ always produce different results.
❑ None of them are correct.

1.9

❑ may produce different results.
❑ never produce different results.
❑ always produce different results.
❑ None of them are correct.

1.10

- ❑ an unnamed procedure.
- ❑ a named procedure.
- ❑ a list of local variables.
- ❑ a list of global variables.

1.11

❑ (cons x y) ❑ (list x y) ❑ (append x y) ❑ None of them

1.12

❑ `'()` ❑ `'(x . y)` ❑ `'(x)` ❑ `'(())`

1.13 ❑ (30 80) ❑ (10 15 10) ❑ (10 15) ❑ (10 10 15)

1.14

- ❑ is a plain list.
- ❑ contains nonnumerical values.
- ❑ contains sublists.
- ❑ is not a pair.

1.15 ❑ 20 ❑ 40 ❑ 25 ❑ (20 40 25)

1.16

❑ a pair. ❑ not a pair. ❑ a string. ❑ `0`

1.17

❑ Call-by-value ❑ Call-by-alias ❑ Call-by-name ❑ Return value

1.18

❑ Call-by-value ❑ Call-by-alias ❑ Call-by-name ❑ Return value

1.19

- ❑ Use multiple return-statements.
- ❑ Split the procedure into multiple procedures
- ❑ Put the values in a pair and return the pair
- ❑ Put the values in a list and return the list

1.20

- ❑ the solutions of the size-1, size-2, ..., size-n problems.
- ❑ a loop variable that is incremented in each iteration.
- ❑ the solutions of the size-n, size-(n−1), size-(n−2), ..., size−1 problems.
- ❑ the solution of the size-(n−1) problem, and finally find the solution of the size-n problem.
- ❑ the solution of the size-n problem based on the hypothetical solution of the size-(n−1) problem.

5.9 Homework, programming exercises, and projects

1. Multiple Choice. Choose only one answer for each question. Choose the best answer if more than one answer is acceptable.

1.1 ❑ C ❑ Lisp ❑ Scheme ❑ Prolog

1.2

❑ Lambda calculus ❑ Predicate logic ❑ Turing machine ❑ Boolean logic

1.3

❑ A fact can be considered a special case of a rule.
❑ A fact can be considered a special case of a goal.
❑ A rule can be considered a special case of a fact.
❑ A rule can be considered a special case of a goal.

1.4

❑ Call-by-value ❑ Call-by-alias ❑ Return value ❑ All of them

1.5 ❑ Use multiple return statements in the predicate.
❑ Use a single return statement to return a list that contains the multiple values required.
❑ Use multiple named variables to hold the values.
❑ Use multiple unnamed variables to hold the values.

1.6 ❑ their predicates are the same. ❑ their arities are the same.
❑ their corresponding arguments match. ❑ all of the above are true.

1.7

❑ the head of the predicate. ❑ the neck of the predicate.
❑ the body of the predicate. ❑ the number of arguments of the predicate.

1.8

❑ a single rule. ❑ a single clause in a rule.
❑ the fact/rule base. ❑ the body of the next rule.

1.9

❑ is an imperative feature that should be discouraged.
❑ will cause a dead loop for every goal.
❑ will cause a dead loop when no match can be found.
❑ will never cause a dead loop.

1.10 ❑ constant. ❑ placeholder. ❑ predicate. ❑ question.

1.11

```
?-member(apple, [orange, apple, pear]).
```

❑ true? ❑ X = apple ❑ [apple, pear] ❑None of them

1.12

```
?- child_of(mary, [amy | T]).
```

❑ T = [amy, david, conrad] ❑ T = [david, conrad]

❑ T = david, conrad ❑ T = conrad

1.13

```
?- child_of(mary, [amy | [H | [conrad]]]).
```

❑ H = [amy] ❑ H = [david] ❑ H = david ❑ H = [amy, conrad]

1.14

```
?- member(X,[81, 25, 9, 29]), Y is X*X, Y<400.
```

❑ X = [81, 25, 9, 29] ❑ X = 9 Y =29 ❑ X = 9 Y = 81 ❑ X = 81 Y =400

1.15 ❑ the last element of a list. ❑ the length of a list.

❑ whether an element is a member of a list. ❑ the sum of all members in a list.

1.16

❑ all existing backtrack points will be removed.

❑ all existing recursive exit points will be removed.

❑ all existing backtrack points and recursive exit points will be removed.

❑ none of the existing backtrack points and recursive exit points will be removed.

1.17

❑ Stop searching immediately.

❑ Remove all existing backtracking points.

❑ Stop search after the first match is found.

❑ Remove all backtracking and recursive exit points.

1.18

❑ Jump to the previous repeat predicate. ❑ Remove one backtracking point.

❑ Return false. ❑ Stop searching immediately.

1.19

❑ fails immediately.

❑ succeeds and adds a backtracking point.

❑ fails at the end of the database.

❑ fails when it is visited (executed) for the second time.

1.20 W

❑ It takes a single character from the keyboard and prints it.

❑ It repeatedly takes a character from the keyboard and prints it.

- ❑ It takes a character from the keyboard, prints it, and exits if the character is a digit.
- ❑ It takes a character from the keyboard, prints it, and exits if the character is NOT printable.

1.21

- ❑ Semantic web performs semantic check of pages submitted to web.
- ❑ Semantic web enables keyword-based search.
- ❑ Semantic web better supports automatic processing and integration of web information.
- ❑ Semantic web is based on Prolog.

1.22

- ❑ Prolog is frequently used to describe the semantic web as a markup language.
- ❑ Prolog is frequently used for writing the parsers of the semantic description language.
- ❑ Prolog is a service-oriented programming language.
- ❑ All Prolog programs are semantic web.

6.9 Homework, programming exercises, and projects

1. Multiple Choice. Choose only one answer for each question. Choose the best answer if more than one answer is acceptable.

1.1

- ❑ imperative
- ❑ object-oriented
- ❑ functional
- ❑ service-oriented

1.2

- ❑ C
- ❑ C++
- ❑ C#
- ❑ None of them

1.3

- ❑ service providers
- ❑ service brokers
- ❑ application builders
- ❑ None of them

1.4

- ❑ service providers
- ❑ service brokers
- ❑ application builders
- ❑ None of them

1.5

- ❑ preprocessor directives
- ❑ macros
- ❑ overriding
- ❑ switch statement

1.6

- ❑ multiple inheritance
- ❑ array
- ❑ pointer
- ❑ for each statement

1.7

- ❑ declaration in C++
- ❑ printf statement
- ❑ pointer
- ❑ header files

1.8

- ❑ C
- ❑ C++
- ❑ C#
- ❑ Scheme

1.9

- ❑ control flow of web services
- ❑ interface of web services
- ❑ syntax of web services
- ❑ semantics of web services

1.10

- ❑ describing the interface of web services
- ❑ composing SOA applications
- ❑ publishing web services
- ❑ calling the remote web services

1.11

- ❑ describing contact information of service providers
- ❑ describing the service types of the web services

❑ describing technical detail for remote invocation of web services

❑ describing testing results on the reliability and trustworthiness of web services

1.12

❑ describing the interface of web service ❑ composing SOA applications

❑ publishing web services ❑ calling remote web services

1.13

❑ A web service is intended for being accessed by a computer program.

❑ A web service is intended for being accessed by a human user.

❑ A web application is intended for being accessed by a computer program.

❑ Web application is a synonym of web service.

1.14

❑ An object with implementation ❑ An object without implementation

❑ A class definition ❑ An constructor

1.15

❑ Add Proxy ❑ Add Endpoint ❑ Add Web ❑ Add Service Reference

1.16

❑ It is identical to SOA software, and there is no difference.

❑ SaaS does not use SOA technology at all.

❑ SaaS is similar to SOA software; however, it is often hosted on a cloud environment.

❑ SaaS is the same as a web service.

1.17

❑ Software components and services ❑ Application development environment

❑ Computing capacity ❑ Memory and disk space

1.18

❑ Structured data ❑ Semi-structured data

❑ Unstructured data ❑ All of these types

1.19

❑ Noise elimination and fault tolerance ❑ Poly-structured data

❑ Real-time data processing ❑ Volatility of data

1.20

❑ Infrastructure ❑ Management ❑ Analytic techniques ❑ Data types

7.8 Homework and programming exercises

1. Multiple Choice. Unless other specified, choose one answer only for each question. Choose the best answer if more than one answer is acceptable.

1.1

❑ compilation ❑ interpretation ❑ two-step translation ❑ None of them

1.2

❑ ABC ❑ C and C++ ❑ Fortran ❑ Prolog

1.3

❑ functional ❑ imperative ❑ logic ❑ object-oriented

1.4

❑ coding simplicity ❑ execution efficiency
❑ static memory instead of dynamic memory ❑ expressivity

1.5

❑ Use indentation ❑ Use curly braces ❑ Use square brackets ❑ Use parentheses

1.6

❑ character type ❑ double type ❑ float type ❑ int type

1.7

❑ int type ❑ float type ❑ string type ❑ character type

1.8

❑ Yes ❑ No

1.9

❑ lambda functions ❑ higher-order functions ❑ Recursive functions ❑ All of them

1.10

❑ To replace if-then-else statement ❑ to replace if-then statement
❑ To check multiple conditions in a sequence ❑ to replace the while loop

1.11

❑ if-then-else ❑ if-then statement ❑ if-elif-else statement ❑ for loop

1.12

❑ Yes ❑ No

1.13

❑ True ❑ False

1.14

❑ prefix notation ❑ infix notation ❑ postfix notation ❑ None of them

1.15 A

❑ True ❑ False

1.16

❑ filter ❑ map ❑ reduce ❑ variadic arguments

1.17

❑ apply an operation to all the elements of a list.
❑ calculate a single value from a list of values.
❑ place a function call in the place where its return value is expected.
❑ use a predicate to select elements that satisfy the predicate.

1.18

❑ apply an operation to all the elements of a list.
❑ calculate a single value from a list of values.
❑ place a function call in the place where its return value is expected.
❑ use a predicate to select elements that satisfy the predicate.

1.19

❑ apply an operation to all the elements of a list.
❑ calculate a single value from a list of values.
❑ place a function call in the place where its return value is expected.
❑ use a predicate to select elements that satisfy the predicate.

1.20

❑ True ❑ False

1.21

❑ True ❑ False

1.22

❑ array ❑ dictionary ❑ list ❑ set ❑ tuple

1.23

❑ array ❑ dictionary ❑ list ❑ set ❑ tuple

1.24

❑ array ❑ dictionary ❑ list ❑ set ❑ tuple

1.25

❑ array ❑ dictionary ❑ list ❑ set ❑ tuple

1.26

❑ ~class name ❑ constructor ❑ __init__ ❑ ~init~

1.27

❑ zero ❑ one only ❑ two or less ❑ multiple

1.28

❑ does not need to put different types of data into a list.
❑ lists can take different types of elements without using polymorphic references.
❑ cannot implement a linked list.
❑ uses value semantics, instead of reference semantics.

1.29

❑ Yes ❑ No

1.30

❑ are standard files.
❑ are not a part of the file system
❑ do not need a file buffer, which is different from the other files.
❑ can process (input and output) string types of data only.

1.31

❑ read ❑ seek ❑ open ❑ close

1.32

❑ Returns the current cursor position.
❑ Changes the file cursor's position.
❑ Writes a string to the file and return the number of characters written.
❑ Clears the file buffer by removing the character left behind by a previous operation.

1.33

❑ Reads n lines from file. ❑ Reads n characters from file.
❑ Reads n integers from file. ❑ Reads n float from file.

1.34

❑ try-catch-finally ❑ catch-try-finally ❑ try-except-finally ❑ except-try-finally

1.35

❑ try ❑ catch ❑ except ❑ finally