Data Structures and Algorithms in C++
(Second Edition)
M. T. Goodrich, R. Tamassia, and D. M. Mount
John Wiley & Sons

Solution of Exercise R-1.1

All are valid except the one containing a $ sign.

Data Structures and Algorithms in C++
(Second Edition)
M. T. Goodrich, R. Tamassia, and D. M. Mount
John Wiley & Sons

Solution of Exercise R-1.3

```
struct Pair {
  int first;
  double second;
};
```

Data Structures and Algorithms in C++
(Second Edition)
M. T. Goodrich, R. Tamassia, and D. M. Mount

John Wiley & Sons

Solution of Exercise R-1.4

After execution, `s` contains `"abcabcdabc"`. The last seven characters, `"abcdabc"`, arises from operation `s` + `t`[1] + `s`, and the first `"abc"` arises from the fact that the assignment uses += to concatenate the contents to `s`.

Data Structures and Algorithms in C++
(Second Edition)
M. T. Goodrich, R. Tamassia, and D. M. Mount
John Wiley & Sons

Solution of Exercise R-1.5

(y + (2 * (z ++))) < (3 - (w / 5)).

Data Structures and Algorithms in C++
(Second Edition)
M. T. Goodrich, R. Tamassia, and D. M. Mount
John Wiley & Sons

Solution of Exercise R-1.6

Each pointer dp[i] points to a variable that first needs to be allocated before being initialized. Once allocated, we need to use *dp[i] to access the double.

```
double* dp[10]
for (int i = 0; i < 10; i++) {
  dp[i] = new double;
  *dp[i] = 0.0;
}
```

Data Structures and Algorithms in C++
(Second Edition)
M. T. Goodrich, R. Tamassia, and D. M. Mount
John Wiley & Sons

Solution of Exercise R-1.7

```
int sumToN(int n) {
  int sum = 0;
  for (int i = 1; i <= n; i++)
    sum += i;
  return sum;
}
```

Data Structures and Algorithms in C++
(Second Edition)
M. T. Goodrich, R. Tamassia, and D. M. Mount
John Wiley & Sons

Solution of Exercise R-1.8

[bool isMultiple(long n, long m) if (n else return false

Data Structures and Algorithms in C++
(Second Edition)
M. T. Goodrich, R. Tamassia, and D. M. Mount
John Wiley & Sons

Solution of Exercise R-1.9

```
void printArray(int** A, int m, int n) {
  for (int i = 0; i < m; i++) {
    for (int j = 0; j < n; j++) {
      std::cout << A[i][j] << ' ';
    }
    std::cout << endl;
  }
}
```

Data Structures and Algorithms in C++
(Second Edition)

M. T. Goodrich, R. Tamassia, and D. M. Mount

John Wiley & Sons

Solution of Exercise R-1.10

Both functions produce the same output. Because its argument is called by reference, the function `g` modifies the contents of its actual argument (by incrementing it). In contrast, the argument to function `f` is passed by value, and hence its value does not change.

Data Structures and Algorithms in C++
(Second Edition)
M. T. Goodrich, R. Tamassia, and D. M. Mount
John Wiley & Sons

Solution of Exercise R-1.12

```
bool CreditCard::chargeIt(double price) {
  if ((price <= 0) || (price + balance > double(limit)))
    return false;                       //price not positive or limit is met
  balance += price;
  return true;                          // the charge goes through
}

void makePayment(double payment) {
  if (payment <= 0) return;             // ignore negative payment
  balance -= payment;
}
```

Data Structures and Algorithms in C++
(Second Edition)
M. T. Goodrich, R. Tamassia, and D. M. Mount
John Wiley & Sons

Solution of Exercise R-1.13

This solution assesses a fixed interest rate. A better solution would involve creating an interest rate member variable, which could be adjusted.

```
void makePayment(double payment) {          // pay with interest
  const double interestRate = 0.10;         // 10 percent interest
  if (payment <= 0) return;                 // ignore negative payment
  balance -= payment * (1 + interestRate);
}
```

Data Structures and Algorithms in C++
(Second Edition)
M. T. Goodrich, R. Tamassia, and D. M. Mount
John Wiley & Sons

Solution of Exercise R-1.14

Processing of dates would involve a number of additional elements. To simplify things, let us assume that there is a special class Date, which has a comparison function isLaterThan. Each payment transaction is provided with two additional arguments, the due date and the payment date. Finally, we assume a fixed late fee of $10.00.

```
void makePayment(
  double payment,                              // payment amount
  const Date& dueDate,                         // payment due date
  const Date& paymentDate)                     // date of payment
{
  const double lateFee = 10.00;                // 10 dollar late fee
  if (payment <= 0) return;                    // ignore negative payment
  balance -= payment;
  if (paymentDate.isLaterThan(dueDate))        // past due?
    balance -= lateFee;
}
```

Data Structures and Algorithms in C++
(Second Edition)
M. T. Goodrich, R. Tamassia, and D. M. Mount

John Wiley & Sons

Solution of Exercise R-1.15

The following functions can be added to the end of the class definition.

```
class CreditCard {
  // ... add these new modifier functions in the public section
  void setNumber(const string& newNumber) { number = newNumber; }
  void setName(const string& newName) { name = newName; }
  void setBalance(double newBalance) { balance = newBalance; }
  void setLimit(int newLimit) { limit = newLimit; }
};
```

Data Structures and Algorithms in C++
(Second Edition)
M. T. Goodrich, R. Tamassia, and D. M. Mount
John Wiley & Sons
Solution of Exercise R-1.16

```
for (int j=1; j <= 58; j++) {
    wallet[0]->chargeIt(double(i));
    wallet[1]->chargeIt(2.0 * i);
    wallet[2]->chargeIt(double(3 * i));
}
```

This change will cause credit card 2 to go over its limit.

Data Structures and Algorithms in C++
(Second Edition)
M. T. Goodrich, R. Tamassia, and D. M. Mount

John Wiley & Sons

Solution of Exercise R-1.17

```
class AllKinds {
private:
  int   intMem;
  long  longMem;
  float floatMem;
public:
  AllKinds() { intMem = 4; longMem = 23L; floatMem = 3.14159F; }
  void setInt(int i) { intMem = i; }
  void setLong(long l) { longMem = l; }
  void setFloat(float f) { floatMem = f; }
  int getInt() const { return intMem; }
  long getLong() const { return longMem; }
  float getFloat() const { return floatMem; }
  long addIntLong() const { return long(intMem) + longMem; }
  float addIntFloat() const { return float(intMem) + floatMem; }
  float addLongFloat() const { return float(longMem) + floatMem; }
  float addAll() const { return float(intMem) + float(longMem) + floatMem; }
};
```

Data Structures and Algorithms in C++
(Second Edition)

M. T. Goodrich, R. Tamassia, and D. M. Mount

John Wiley & Sons

Solution of Exercise R-1.18

```
bool isMultiple(long n, long m) {
    if (n % m == 0)
        return true;
    else
        return false;
}
```

Data Structures and Algorithms in C++
(Second Edition)
M. T. Goodrich, R. Tamassia, and D. M. Mount
John Wiley & Sons

Solution of Exercise R-1.19

The following remarkably short and tricky function determines whether a nonnegative integer i is a power of 2.

```
bool isTwoPower(int i) {
  return (i != 0) && (((i-1) & i) == 0);
}
```

The function makes use of the fact that the binary representation of a of $i = 2^k$ is a 1 bit followed by k 0 bits. In this case, the binary representation of $i - 1$ is a 0 bit followed by i 1 bits. Thus, when we take the bitwise "and" of the two bit strings, all the bits cancel out.

$$\begin{aligned} i = 1024_{10} &= 0000100000000000_2 \\ i - 1 = 1023_{10} &= 0000011111111111_2 \\ i \mathbin{\&} (i-1) &= 0000000000000000_2 \end{aligned}$$

If i is not a power of 2 and $i > 0$, then the bit strings for i and $i - 1$ share at least one bit in common, the highest order bit, and so their bitwise "and" will be nonzero. We need to include a special check for zero, since it will pass this test but zero is not a power of 2.

Data Structures and Algorithms in C++
(Second Edition)
M. T. Goodrich, R. Tamassia, and D. M. Mount
John Wiley & Sons

Solution of Exercise R-1.20

Here is a solution based on the use of a for loop.

```
int sumToN(int n) {
   int sum = 0;
   for (int i = 1; i < n; i++) sum += i;
   return sum;
}
```

Here is a different solution, based instead on recursion.

```
int sumToN(int n) {
   if (n <= 0)
      return 0;
   else
      return (n-1 + sumToN(n-1));
}
```

Data Structures and Algorithms in C++
(Second Edition)
M. T. Goodrich, R. Tamassia, and D. M. Mount
John Wiley & Sons

Solution of Exercise R-1.21

Here is a solution based on the use of a for loop.

```
int sumOdd(int n) {
   int sum = 0;
   for (int i = 1; i < n; i+=2) sum += i;
   return sum;
}
```

Data Structures and Algorithms in C++
(Second Edition)

M. T. Goodrich, R. Tamassia, and D. M. Mount

John Wiley & Sons

Solution of Exercise R-1.22

```
int divideByTwo(double x) {
    int result = 0;
    while (x >= 2) {
        x = x/2;
        result++;
    }
    return result;
}
```

Data Structures and Algorithms in C++
(Second Edition)
M. T. Goodrich, R. Tamassia, and D. M. Mount
John Wiley & Sons

Solution of Exercise C-1.3

```
bool allDistinct(const vector<int>& a) {
  for (int i = 0; i < a.size()−1; i++) {
    for (int j = i+1; j < a.size(); j++) {
      if (a[i] == a[j]) return false;
    }
  }
  return true;
}
```

Data Structures and Algorithms in C++
(Second Edition)
M. T. Goodrich, R. Tamassia, and D. M. Mount
John Wiley & Sons

Solution of Exercise C-1.4

```
void printOddInts(const vector<int>& v) {
  for( int i = 0; i < v.size(); i++ ) {
    if( v[i] % 2 == 1 ) {                    // check (v mod 2) == 1
      cout << v[i] << endl;
    }
  }
}
```

Data Structures and Algorithms in C++
(Second Edition)
M. T. Goodrich, R. Tamassia, and D. M. Mount

John Wiley & Sons

Solution of Exercise C-1.5

Our solution represents the array as an STL vector v for convenience, and this solution works for vectors of any size, not just 52. For i running from the index of the last element of v down to 1, a random integer r ranging from 0 to i is generated. The ith element of v is swapped with the rth element. The function rand returns a random integer, and we function use the fact that rand()%(i+1) generates a random number between 0 and i.

```
#include <cstdlib>                              // needed for rand()
// ...
void shuffle(vector<int>& v) {
  for(int i = v.size()−1; i > 0; i−−) {         // work from back to front
    int r = rand() % (i+1);                     // random int from 0 to i
    int temp = v[i];                            // swap v[i] with v[r]
    v[i] = v[r];
    v[r] = temp;
  }
}
```

Data Structures and Algorithms in C++
(Second Edition)

M. T. Goodrich, R. Tamassia, and D. M. Mount

John Wiley & Sons

Solution of Exercise C-1.6

The algorithm operates recursively. We maintain two lists, bag, which holds the characters that have not yet been put into the permutation and permutation, which holds the characters that are in the permutation. Initially the bag list has all the characters and the permutation list is empty. One by one, characters are copied from the bag to the permutation and back again. When all characters of the bag have been added to the permutation, we print the result. If characters remain in the bag, we process them as follows. We iterate through each character of the current bag, remove this character and add it to the rear of the permutation. Then we recursively repeat the process on the remaining bag. On returning to the recursive calls, we remove the last element from the permutation, and put it back in the bag. In this way, each element of the bag takes turns being the last element of the permutation. We have omitted the header material, which includes the files <cstdlib>, <list>, <string>, and <iostream>.

```
using namespace std;

ostream& operator<<(ostream& out, const list<char>& L) {
  list<char>::const_iterator p = L.begin();
  while (p != L.end()) { out << *p; p++; }
  return out;
}

void permute(list<char>& bag, list<char>& permutation) {
  if(bag.empty())                              // empty bad means we're done
    cout << permutation << endl;
  else {
    list<char>::iterator p = bag.begin();      // for each element left in bag
    while (p != bag.end()) {
      list<char>::iterator n = p; n++;         // save next position
      char c = *p;                             // remove next item from bag
      bag.erase(p);
      permutation.push_back(c);                // add to back of permutation
      permute(bag, permutation);               // recursively permute the rest
      permutation.pop_back();                  // remove the last element
      bag.insert(n, c);                        // ... and restore to the bag
      p++;
    }
  }
}

void printPermutations(list<char>& elements) {
  list<char> bag = elements;
  list<char> permutation;
  permute(bag, permutation);
}

main()
{
  const char elts[] = { 'a', 'b', 'c', 'd', 'e', 'f' };
  list<char> elements;
  for (int i = 0; i < 6; i++) elements.push_back(elts[i]);
  printPermutations(elements);
  return EXIT_SUCCESS;
}
```

Data Structures and Algorithms in C++
(Second Edition)

M. T. Goodrich, R. Tamassia, and D. M. Mount

John Wiley & Sons

Solution of Exercise C-1.7

The following program reads lines from the input stream, and appends each line (as a string) to the back of an STL vector. It then pops the contents of the vector, and prints the resulting lines.

```
int main() {
  vector<string> stack;
  while (!cin.eof()) {                    // read until end of file
    char c;
    string line;
    do {                                  // read a line
      c = cin.get();                      // read a character
      if (cin.eof()) break;               // break if end of file
      line.push_back(c);                  // append c to string
    } while (c != '\n');
    stack.push_back(line);                // append line to end
  }
  while (!stack.empty()) {                // pop lines and print
    cout << stack.back();
    stack.pop_back();
  }
  return EXIT_SUCCESS;
}
```

Data Structures and Algorithms in C++
(Second Edition)
M. T. Goodrich, R. Tamassia, and D. M. Mount
John Wiley & Sons

Solution of Exercise C-1.8

```
typedef vector<double> DblVec;
DblVec product(const DblVec& a, const DblVec& b ) {
    DblVec c = DblVec(a.size());
    for(int i = 0; i < a.size(); i++) {
      c[i] = a[i] * b[i];
    }
    return c;
}
```

Data Structures and Algorithms in C++
(Second Edition)
M. T. Goodrich, R. Tamassia, and D. M. Mount
John Wiley & Sons

Solution of Exercise C-1.9

C++ provides two methods to overload operators. The method we show below translates the operation $u+v$ into the member function call u.operator+ (v). It then invokes the member function for u with v as the argument.

```
class Vector2 {
private:
        double x, y;
public:
        Vector2(double _x, double _y) // constructor
          : x(_x), y(_y) { }
        Vector2 operator+(const Vector2& v) // vector addition
          { return Vector2(x + v.x, y + v.y); }
        Vector2 operator*(double s) // scalar-vector multiplication
          { return Vector2(s*x, s*y); }
        double operator*(const Vector2& v) // dot product
          { return x*v.x + y*v.y; }
};
```

Data Structures and Algorithms in C++
(Second Edition)
M. T. Goodrich, R. Tamassia, and D. M. Mount

John Wiley & Sons

Solution of Exercise C-1.10

If we express i as the sum of powers of 2, for example, consider the case $i = 11$. We have

$$i = 11_{10} = 1011_2 = 1 + 2 + 8 = 2^0 + 2^1 + 2^3.$$

We observe that in this case,

$$2^i = 2^{(2^0+2^1+2^3)} = (2^{2^0}) \cdot (2^{2^1}) \cdot (2^{2^3}).$$

Our program maintains two variables, repeatedSquare, which holds the value 2^{2^j} and partialProduct, which holds the value of the above product. To determine whether the binary expansion of i contains a 1 bit at a certain position, we perform the bitwise-or of this number with 1, and then shift right by one position. If the extracted bit is 1, then we augment the partial product by multiplying with the current repeated square value, and in any case, we square the repeated square value.

```
long twoToThe( int i ) {
  long partialProduct = 1;
  long repeatedSquare = 2;

  while( i != 0 ) {
    int bit = (i & 1);
    i = i >> 1;
    if( bit == 1 ) {
      partialProduct *= repeatedSquare;
    }
    repeatedSquare *= repeatedSquare;
  }
  return partialProduct;
}
```

Data Structures and Algorithms in C++
(Second Edition)
M. T. Goodrich, R. Tamassia, and D. M. Mount
John Wiley & Sons

Solution of Exercise C-1.11

```
int GCD(int n, int m) {
  int dividend = n;
  int divisor = m;
  int remainder = 1;
  while (remainder > 0) {
    remainder = dividend % divisor;
    dividend = divisor;
    divisor = remainder;
  }
  return dividend;
}
```