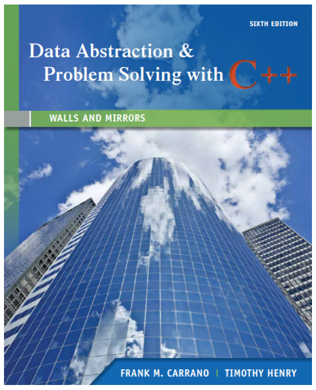

SIXTH EDITION
Data Abstraction & Problem Solving with C++
WALLS AND MIRRORS
FRANK M. CARRANO | TIMOTHY HENRY

Data Abstraction & Problem Solving with C++ Walls and Mirrors (6th edition)

by

Frank M. Carrano

University of Rhode Island

and

Timothy M. Henry

New England Institute of Technology

Connect with us

making it real Frank M. Carrano

Errata List

Last updated on June 4, 2015.
The date after each error is the date it was posted on this list. Subsequent printings of the book will correct these errors.

If you find an error and it is not on this list, please e-mail a description of the error, the page number, and the tile/edition of the book.

Chapter 1

Page 23 (Nov. 20, 2012)
In Listing 1-1, add the statement

```
using namespace std;
```

after the compiler directives.

Page 26 (Jan. 7, 2015)
In the 7th item of the Chapter Summary, add an "s" to the first word of the second sentence:

Modules should be loosely coupled.

C++ Interlude 1

Page 38 (Nov. 20, 2012)
In Listing C1-3, delete the semicolon in the `template` statement:

template<class ItemType>~~;~~ // Indicates this is a template definition

Page 38 (Feb. 26, 2015)
Right after Listing C1-3, delete the semicolon in the `template` statement:

template<class ItemType>~~;~~

Page 39 (Feb. 26, 2015)
Near the top of the page, delete the semicolon in the `template` statement:

template<class ItemType>~~;~~

Page 39 (Nov. 20, 2012)
In Listing C1-4, delete the semicolon in the four `template` statements:

template<class ItemType>~~;~~

Pages 41 and 42 (Jan. 17, 2013)
Although the code in Listing C1-6 works, each constructor calls the `PlainBox` constructor twice, once explcitly and once implicitly. A detailed discussion of defining constructors in a derived class is at this link.

Page 43 (Jan. 17, 2013)
In Listing C1-8, delete the explicit call to the `PlainBox` constructor, since it is called implicitly. See the previous note for pages 41 and 42.

Page 45 (Nov. 20, 2012)

Replace the declaration of the method `setItem` in the 4th line of displayed code in the top half of the page with

```
virtual void setItem(const ItemType& theItem);
```

Chapter 2

Page 59 (Feb. 26, 2015)
Insert a cast into the first statement of the method `writeBackward` at the top of the page, as follows:

```
int length = (int)s.size(); // Length of string
```

Page 68 (May 14, 2015)
In the last quarter of the page, the relational signs are missing between the subscripted variables:
The line should be:

```
anArray[0] <= anArray[1] <= anArray[2] <= . . . <= anArray[size - 1]
```

Page 70 (May 14, 2015)
In the last quarter of the page, the relational signs are missing between the subscripted variables and `target`:
The line should be:

```
anArray[first] <= target <= anArray[last]
```

Page 72 (Nov. 10, 2012)
Checkpoint Question 7 should be numbered as 8.

Page 74 (Feb. 26, 2015)
In the last margin note, the "less than or equal to" operator is missing:

Partition `anArray`
into three parts:
values <= p, p, and
values >= p

Page 79 (Nov. 10, 2012)
Checkpoint Question 8 should be numbered as 9.

Page 84 (Nov. 20, 2012)
Revise the definition of the function `getNumberOfGroups` as follows:

- In the second `return` statement, delete the >; the statement should be

  ```
  return 0;
  ```

- In the third `return` statement, replace the two calls to `g` with calls to `getNumberOfGroups`; the statement should be

  ```
  return getNumberOfGroups(n - 1, k - 1) + getNumberOfGroups(n - 1, k);
  ```

Checkpoint Question 9 should be numbered as 10.

Page 87 (Feb. 26, 2015)
Insert a cast into the first statement of both versions of the method `writeBackward`, as follows:

```
int length = (int)s.size(); // Length of string
```

Page 87 (Nov. 10, 2012)
Checkpoint Question 10 should be numbered as 11.

Chapter 3

Page 103 (Nov. 10, 2012)
Revise Checkpoint Question 2 as follows:

If a client of `ArrayBag` creates a bag `aBag` and a vector `v` containing five items, what happens to those items after the statement `v = aBag.toVector()` executes?

C++ Interlude 2

Page 120 (Nov. 10, 2012)
The first paragraph of Section C2.3 defines the *free store*. Many people call the free store a heap. Because Chapter 17 introduces the ADT heap—and the free store is not an ADT heap—we offer the term "application heap" instead of "heap" as an alternate to the free store. This book typically uses "free store" to avoid confusion with the ADT heap.

Therefore, replace the phrase "called the **heap**, or **free store**," in the fifth line of this first paragraph with "called the **free store**, or **application heap**,"

Page 123 (May 21, 2015)
In the third line from the bottom of the page, delete `RED` in `ToyBox<double>(~~RED~~)`.

Page 131 (Nov. 20, 2012)
The section number 2.5.1 should be C2.5.1.

Chapter 4

Page 146 (Nov. 20, 2012)
In the method `clear` at the top of the page, the statement

```
nodeToDeletePtr = nullptr;
```

near the end of the method will be flagged by the compiler because `nodeToDeletePtr` is undefined after the `while` statement. The simple fix is to delete this statement; afterall, `nodeToDeletePtr` is local to the method and will no longer exist after the method ends execution. A better solution is to move the declaration of `nodeToDeletePtr` to before the `while` statement , as follows:

```
template<class ItemType>     <-------The first two lines are on page 145.
void LinkedBag::clear()      <-------
{
   Node<ItemType>* nodeToDeletePtr = headPtr;
   while (headPtr != nullptr)
   {
      headPtr = headPtr->getNext();

      // Return node to the system
      nodeToDeletePtr->setNext(nullptr);
      delete nodeToDeletePtr;
      nodeToDeletePtr = headPtr;
   } // end while
   // headPtr is nullptr; nodeToDeletePtr is nullptr

   itemCount = 0;
} // end clear
```

Page 147 (Nov. 20, 2012, Mar. 19, 2013, Nov. 16, 2013)
In the copy constructor at the bottom of the page,

- In the 4th line, replace -> with a period:

  ```
  itemCount = aBag.itemCount;
  ```

- In the 5th line, replace -> with a period, and add a semicolon at the end:

  ```
  Node<ItemType>* origChainPtr = aBag.headPtr;
  ```

- In the `while` statement, replace `origPtr` with `origChainPtr`:

  ```
  while (origChainPtr != nullptr)
  ```

- The first statement in the body of the `while` loop,

  ```
  origChainPtr = origChainPtr->getNext(); // Advance pointer
  ```

  is misplaced. It should appear just before the `while` statement and again on page 148 as the last statement in the loop's body.

Page 148 (Nov. 20, 2012)
In Section 4.3.1,

- Add a semicolon at the end of the declaration of `fillVector`:

  ```
  void fillVector(vector<ItemType>& bagContents, Node<ItemType>* curPtr) const;
  ```

- In the last line on the page, `LinkedBag<ItemType>::` is missing before the method's name:

  ```
  void LinkedBag<ItemType>::fillVector(vector<ItemType>& bagContents, Node<ItemType>*
  curPtr) const
  ```

Page 149 (Nov. 20, 2012)

- Near the top of the page, the comment `// end toVector` should be `// end fillVector`
- In the definition of the method `getPointerTo` at the bottom of the page, `LinkedBag<ItemType>::` is missing before the method's name:
  ```
  Node<ItemType>* LinkedBag<ItemType>::getPointerTo(const ItemType& target,
                                                    Node<ItemType>* curPtr) const
  ```

Page 151 (Mar. 4, 2015)

In Listing 4-4,

- Revise the method `displayBag` by replacing the two statements after the `cout` statement with

  ```
  vector<string> bagItems = bag->toVector();
  ```

- Revise the method `main` at the bottom of the page by deleting the semicolon at the end of the first part of the `cout` statement:

  cout << "Enter 'A' to test the array-based implementation\n"~~;~~

Page 152 (Mar. 4, 2015)

Note that the compiler will issue a warning message regarding the `delete` operator in the `main` method. This occurs because `BagInterface`, as given in Listing 1-1 of Chapter 1, is abstract and has a non-virtual destructor. To avoid the warning, you should add the following declaration of a destructor to the definition of `BagInterface`:

```
virtual ~BagInterface() {}
```

This destructor has an empty body; it is not a pure virtual method.

Chapter 5

Page 170 (Nov. 10, 2012)

In Checkpoint Question 3, an initial minus sign is missing. The prefix expression should be
- - *a* / *b* + *c* * *d e f*

Page 171 (Oct. 25, 2013)

The last line of the pseudocode for the method `convert` should be
`return postExp`

Page 181 (Nov. 20, 2012)

In the private section, the declaration of the method `placeQueens` should begin with `bool` instead of `const`:

```
bool placeQueens(Queen* queenPtr);
```

Page 186 (Feb. 25, 2013)
In Exercise 1a, `isPal` should be `isPalindrome`

Page 187 (June 4, 2015)
In Exercise 6, the second statement in the grammar should be

<dot> = •

Page 188 (June 4, 2015)
In Exercise 15, the operator in the `if` statemnt should be >=

Chapter 6

Page 200 (Nov. 10, 2012)
Our notation for a newly created stack in the axioms can cause confusion with C++ syntax when it is used later in method calls. For example, the axiom pseudocode
`(new Stack()).isEmpty()`
would be incorrect as a C++ expression. In C++, it should use -> instead of a dot as follows:
`(new Stack())->isEmpty()`
Therefore, we are deleting the `new` in the notation and making the following two changes near the bottom of the page:

> We can state this axiom succinctly in terms
> of the ADT stack operations as follows, if we represent a newly created stack by the pseudocode expression ~~new~~ `Stack()`:
> (~~new~~ `Stack()).isEmpty() = true`

Page 201 (Nov. 10, 2012)
Make the following changes to the axiom notation in the Note at the top of the page. See the explanation just given for page 200.

> **Note: Axioms for the ADT stack**
> (~~new~~ `Stack()).isEmpty() = true`
> (~~new~~ `Stack()).pop() = false`
> (~~new~~ `Stack()).peek() = error`

Page 220 (Mar. 5, 2015)
In Exercise 15, delete the five occurrences of "new." See the explanation just given for page 200.

Chapter 7

Page 247 (Mar. 19, 2013)

- In the 2nd line, replace -> with a period:

  ```
  Node<ItemType>* origChainPtr = aStack.topPtr;
  ```

- In the 4th line, replace "bag" with "stack" in the comment:

  ```
  topPtr = nullptr; // Original stack is empty;
  ```

- The first statement in the body of the `while` loop,

  ```
  origChainPtr = origChainPtr->getNext();
  ```

  is misplaced. It and its preceding comment should appear just before the `while` statement and again at the end of the loop's body.

Chapter 8

Page 258 (Nov. 10, 2012)
Make the following changes to the axiom notation. See the explanation given previously for page 200 in Chapter 6.

In Line 6: `(`~~`new `~~`List()).isEmpty() = true`

In the Note:

> **Note: Axioms for the ADT list**
> 1. `(`~~`new `~~`List()).isEmpty() = true`
> 2. `(`~~`new `~~`List()).getLength() = 0`
> .
> .
> .
> 6. `(`~~`new `~~`List()).remove(i) = false`
> .
> 8. `(`~~`new `~~`List()).getEntry(i) = error`
> .
> .
> .
> 12. `(`~~`new `~~`List()).setEntry(i, x) = error`

Chapter 9

Page 274 (Mar. 19, 2013)
Delete the sentence that follows the default constructor, as we will need a copy constructor.

~~This is the only constructor we will need.~~

Chapter 10

Page 297 (Oct. 25, 2013)
In Figure 10-3b, the curve for n log n is missing. Here is a revised graph.

Chapter 11

Pages 307, 308, 310, 312, 316, 317, 324 (Mar. 31, 2015)
Each function header on these pages must be preceded by

```
template<class ItemType>
```

Page 313 (Nov. 20, 2012)
In the code at the top of the page, the statement that decrements `loc` is misplaced, and the comments that end the `while` and `for` loops are interchanged. The corrections follow:

```
      while ((loc > 0) && (theArray[loc - 1] > nextItem))
      {
         // Shift theArray[loc - 1] to the right
         theArray[loc] = theArray[loc - 1];
         loc--;
      }  // end while
      // At this point, theArray[loc] is where nextItem belongs
      theArray[loc] = nextItem; // Insert nextItem into sorted region
      ~~loc--;~~
   }  // end for
}  // end insertionSort
```

Page 323 (Nov. 10, 2012)
In the pseudocode for the `partition` algorithm, delete the first line in the body of the `if` statement:

```
if (indexFromLeft < indexFromRight)
{
   ~~Move theArray[firstUnknown] into S1~~
   Interchange theArray[indexFromLeft] and theArray[indexFromRight]
   . . .
```

Page 332 (Nov. 20, 2012)
In Programming Problem 8, replace `DataItem` within the second `for` loop of the function `shellSort` with `ItemType`.

```
for (int unsorted = h; unsorted < n; unsorted++)
{
   ItemType nextItem = theArray[unsorted];
```

C++ Interlude 4

Page 335 (Nov. 20, 2012)
In Listing C4-1, delete the semicolon in the `template` statement:

```
template<class ItemType>~~;~~
```

Page 336 (Nov. 20, 2012)
In Listing C4-2, delete the semicolon in the `template` statement:

```
template<class ItemType>~~;~~
```

Page 338 (Nov. 20, 2012)
Replace the declaration of the method `setItem` in the class `PlainBox` with

```
virtual void setItem(const ItemType& theItem);
```

Page 339 (Nov. 20, 2012)
In the body of the method `setItem`, precede `item` with `PlainBox<ItemType>::`. The statement is then

```
PlainBox<ItemType>::item = theItem; // item has protected access
```

Chapter 12

Page 351 (Feb. 26, 2013)
In `SortedListInterface`, the method `getPosition` should be a `const` method. Thus, its declaration should be

```
virtual int getPosition(const ItemType& anEntry) const = 0;
```

Page 354 (Mar. 19, 2013)
In Section 12.2.2,

- Add the following statement as the last one in the copy constructor:

  ```
  itemCount = aList.itemCount;
  ```

- In the method `copyChain`, delete the two statements that involve `itemCount`.

- In the `else` clause in the method `copyChain`, delete `Node<ItemType>*` that precedes `copiedChainPtr`, since it has been declared already.

Page 360 (May 14, 2015)
The end of the first paragraph should reference Checkpoint Question 12 instead of 11.

Page 360 (Nov. 10, 2012)
We changed the order of the Checkpoint Questions on this page in the first printing: Question 12 is now Question 9, and Questions 9 through 11 are renumbered as 10 through 12. Thus, we have

Checkpoint Question 9 asks you to define `getPosition`. You can do so by using the methods `getLength` and `getEntry`.

Question 9 Define the method `getPosition` for the class `SortedListHasA`.
Question 10 Repeat Checkpoint Question 7 using the method `insertSorted` of the class `SortedListHasA`.
Question 11 Can a client of `SortedListHasA` invoke the method `insert` of the ADT list? Explain.
Question 12 Define the method `removeSorted` for the class `SortedListHasA`.

Note that the online answers assume these changes.

Chapter 13

Page 390 (Nov. 10, 2012)
Part *a* of Checkpoint Question 7 has a "6" before "deli" that should be a blank space. Thus, "a6deli" should be "a deli"

Page 391 (Feb. 25, 2013)
In Exercise 3, the two calls to `peek` should be calls to `peekFront`.

Page 392 (Nov. 10, 2012)
In Exercise 14, delete "`new`" in the list of axioms (see the explanation given previously for page 200 in Chapter 6):

```
(~~new ~~Queue()).isEmpty() = true
(~~new ~~Queue()).dequeue() = false
(~~new ~~Queue()).peekFront() = error
((~~new ~~Queue()).enqueue(item)).dequeue() = ~~new ~~Queue()
((~~new ~~Queue()).enqueue(item)).peekFront() = item
```

Chapter 14

Page 400 (Mar. 19, 2013)
Revise the 4th sentence of the first paragraph, as follows:

The copy constructor uses an initializer, `listPtr(aQueue.listPtr)`, to ~~invoke the list's copy constructor~~ make a shallow copy.

The initializer simply copies the value of `aQueue.listPtr` to `listPtr`.

Page 413 (Feb. 25, 2013)
In exercise 8, change the two occurrences of "front" to "back," and change the two occurrences of "back" to "front." The exercise should read as follows:

The class `ListQueue`, as given in Listing 14-1, maintains the queue's back at the end of a list of the queue's entries and has the front of the queue at the beginning of that list. Note that the list is an object of the class `LinkedList`. What is the impact on the efficiency of the operations `enqueue` and `dequeue` if we were to maintain the queue's back at the beginning of the list and the queue's front at the list's end?

Chapter 15

Page 441 (Nov. 20, 2012)
In Listing 15-1, add the following statement between the `#define` and `template` statements:

```
#include "NotFoundException.h"
```

Page 442 (Nov. 20, 2012)
In the comments that precede the methods `add` and `remove`, delete the `*/` at the end of the `@param` lines.

Chapter 16

Page 460 (Nov. 20, 2012)
In Listing 16-3, replace `void` in the class header with `class`:

```
class BinaryNodeTree : public BinaryTreeInterface<ItemType>
```

Page 463 (Jan. 7, 2015
In the definition of the copy constructor about 2/3 down the page, replace `treePtr` two times with `tree` to be consistent with the header file on page 461:

```
template<class ItemType>
BinaryNodeTree::
          BinaryNodeTree(const BinaryNodeTree& tree)
{
   rootPtr = copyTree(tree.rootPtr);
} // end copy constructor
```

Page 478 (Apr. 24, 2013)
Near the bottom of the page, the last three statements within the innermost `else` clause should be aligned with their previous `else`:

```
else if (N has only one child C)
{
   // C replaces N as the child of Nâ€™ s parent
   if (C is a left child)
      nodeToConnectPtr = nodePtr->getLeftChildPtr()
   else
      nodeToConnectPtr = nodePtr->getRightChildPtr()

   delete nodePtr
   nodePtr = nullptr
   return nodeToConnectPtr
}
```

```
else // N has two children
{ . . .
```

Note that if this was actual C++ code, the misalignment would not change the logic of the code.

Pages 484 - 485 (Apr. 2, 2015)
Make the following changes to the method `readFullTree` at the bottom of page 484:

- Add `treePtr` as the first parameter of the method.
- Move the two statements at the bottom of the page (and the preceding comment) to page 485 between the constructions of the left and right subtrees.

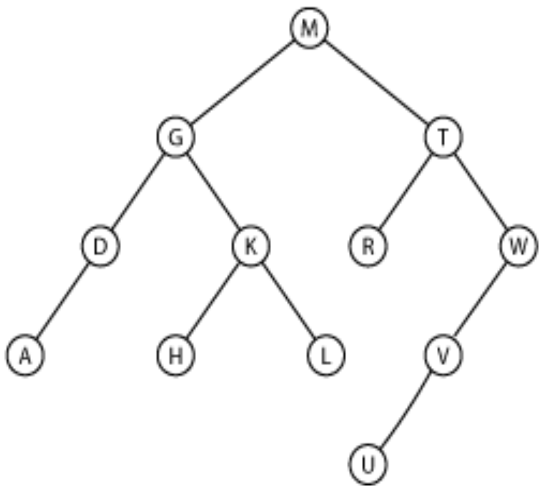
Thus, the method should be as follows:

```
// Builds a full binary search tree from n sorted values in a file.
// Returns a pointer to the treeâ€™s root.
readFullTree(treePtr: BinaryNodePointer, n: integer): BinaryNodePointer

   if (n > 0)
   {
      treePtr = pointer to new node with nullptr as its child pointers

      // Construct the left subtree
      leftPtr = readFullTree(treePtr->getLeftChildPtr(), n / 2)
      treePtr->setLeftChildPtr(leftPtr)

      // Get the root
      rootItem = next item from file
      treePtr->setItem(rootItem)

      // Construct the right subtree
      rightPtr = readFullTree(treePtr->getRightChildPtr(), n / 2)
      treePtr->setRightChildPtr(rightPtr)
      return treePtr
   }
   else
      return nullptr
```

Page 486 (Nov. 5, 2013)
Make the same changes to the method `readTree` that you made to the method `readFullTree` on pages 484 - 485. Also, change the recursive calls so that they are to `readTree` instead of to `readFullTree`. Thus, the method should be as follows:

```
// Builds a full binary search tree from n sorted values in a file.
// Returns a pointer to the treeâ€™s root.
readTree(treePtr: BinaryNodePointer, n: integer): BinaryNodePointer

   if (n > 0)
   {
      treePtr = pointer to new node with nullptr as its child pointers

      // Construct the left subtree
      leftPtr = readTree(treePtr->getLeftChildPtr(), n / 2)
      treePtr->setLeftChildPtr(leftPtr)

      // Get the root
      rootItem = next item from file
      treePtr->setItem(rootItem)

      // Construct the right subtree
      rightPtr = readTree(treePtr->getRightChildPtr(), (n - 1) / 2)
      treePtr->setRightChildPtr(rightPtr)
      return treePtr
   }
   else
      return nullptr
```

Page 489 (Mar. 1, 2013)
Exercise 4 references the binary tree in Figure 15-19 of Chapter 15. This tree is not a binary search tree, as required by this exercise. Instead, begin with the following binary search tree:



Page 490 (Feb. 28, 2013)
In Exercise 7b, the second and third calls to `add` should be calls to `remove`. Thus, the four C++ statements should be

```
     bst.add("Doug");
     bst.remove("Nancy");
     bst.remove("Bob");
     bst.add("Sarah");
```

In Exercise 7c, use Figure 15-14c instead of Figure 15-14b.

C++ Interlude 6

Page 496 (Dec. 5, 2012)
The code snippet mid-page and before the Note has two glitches. The corrections are shown below in red:

```
LinkedIterator<string> currentIterator = myList.begin();
while (currentIterator != myList.end())
{
   cout << *currentIterator; // O(1) operation
   ++currentIterator;
}  // end while
```

Page 497 (Dec. 5, 2012)
The comment at the bottom of this page should end in `*/` instead of `* /`
Note that the available source code is correct.

Page 500 (Dec. 5, 2012)
Each of the functions `count` and `distance` returns a value that is of type `long` instead of `int`. Thus, in the examples of their use, `aceCount` and `numberRemaining` should be declared as `long` instead of `int`.

Chapter 17

Page 511 (Nov. 5, 2013)
In the last line of pseudocode at the bottom of the page, change < to <=:

```
if (items[newDataIndex] <= items[parentIndex])
```

Page 515 (Nov. 11, 2013)
Correct the second paragraph as follows:

Actually, you can replace `itemCount` ~~-1~~ with `itemCount / 2` in the previous `for` statement. Thus, our pseudocode becomes

```
for (index = itemCount / 2 - 1 down to 0)
```

Page 515 (Jan. 10, 2015)
Correct the last line of text to match the logic of the previously corrected pseudocode:

. . . Since the array contains six items,
`index` in the `for` statement begins at 6 / 2 - 1, or 2.

Chapter 18

Page 537 (Dec. 13, 2012)
In listing 18-4, delete the second constructor: `TreeDictionary(int maxNumberOfEntries);`

Page 560 (Dec. 5, 2012)
At the end of the page, exchange the declarations of the two parameters of the method `add`:

```
add(searchKey: KeyType, newItem: ItemType): boolean
```

Page 561 (Dec. 5, 2012)
In the C++ definition of the method `add`, exchange the declarations of the two parameters:

```
bool HashedDictionary<KeyType, ItemType>::add(const KeyType& searchKey,
                                               const ItemType& newItem)
```

Chapter 19

Page 571 (Dec. 5, 2012)
In Listing 19-1, delete the declarations for the two parameterized constructors.

Page 572 (Dec. 5, 2012)
At the top of the page (at the end of Listing 19-1), add `.cpp` after `TriNode` in the `#include` statement.
`#include "TriNode.cpp"`

Page 585 (Nov. 10, 2012)
Checkpoint Question 4 should reference Checkpoint Question 2 instead of 1.

Chapter 21

Page 650 (Nov. 10, 2012)
To clarify the pseudocode for `getItem`, insert `data.getRecord(k)` within the body of the third (innermost) `if` statement, as indicated:
Find data record `data.getRecord(k)` *whose search key equals* `searchKey`

Also, the indentation of various lines is incorrect. Click here to see the correct pseudocode for `getItem`.

Appendix E

Page 757 (Oct. 25, 2013)
In the Inductive Conclusion of Example 3, replace 2k + 1 with $2^{k + 1}$ in the last line of the equations.

(End of errata)