Chapter 10 Some sample B-- programs

```
----------------------------------------------------------------
Programs/format/program.in
----------------------------------------------------------------
printf( "decimal %d, char %c, string %s.\n", 12, '%', "hello" );
```

```
----------------------------------------------------------------
Programs/format/interp.out
----------------------------------------------------------------
decimal 12, char %, string hello.
```

```
----------------------------------------------------------------
Programs/fact/program.in
----------------------------------------------------------------
int fact( int n; )
     begin
          if n == 0 then
               return 1;
          else
               return n * fact( n - 1 );
          end
     end
int i;
for i = 0; i < 10; i = i + 1 do
     printf( "%d factorial = %d\n", i, fact( i ) );
end
```

```
----------------------------------------------------------------
Programs/fact/interp.out
----------------------------------------------------------------
0 factorial = 1
1 factorial = 1
2 factorial = 2
3 factorial = 6
4 factorial = 24
5 factorial = 120
6 factorial = 720
7 factorial = 5040
8 factorial = 40320
9 factorial = 362880
```

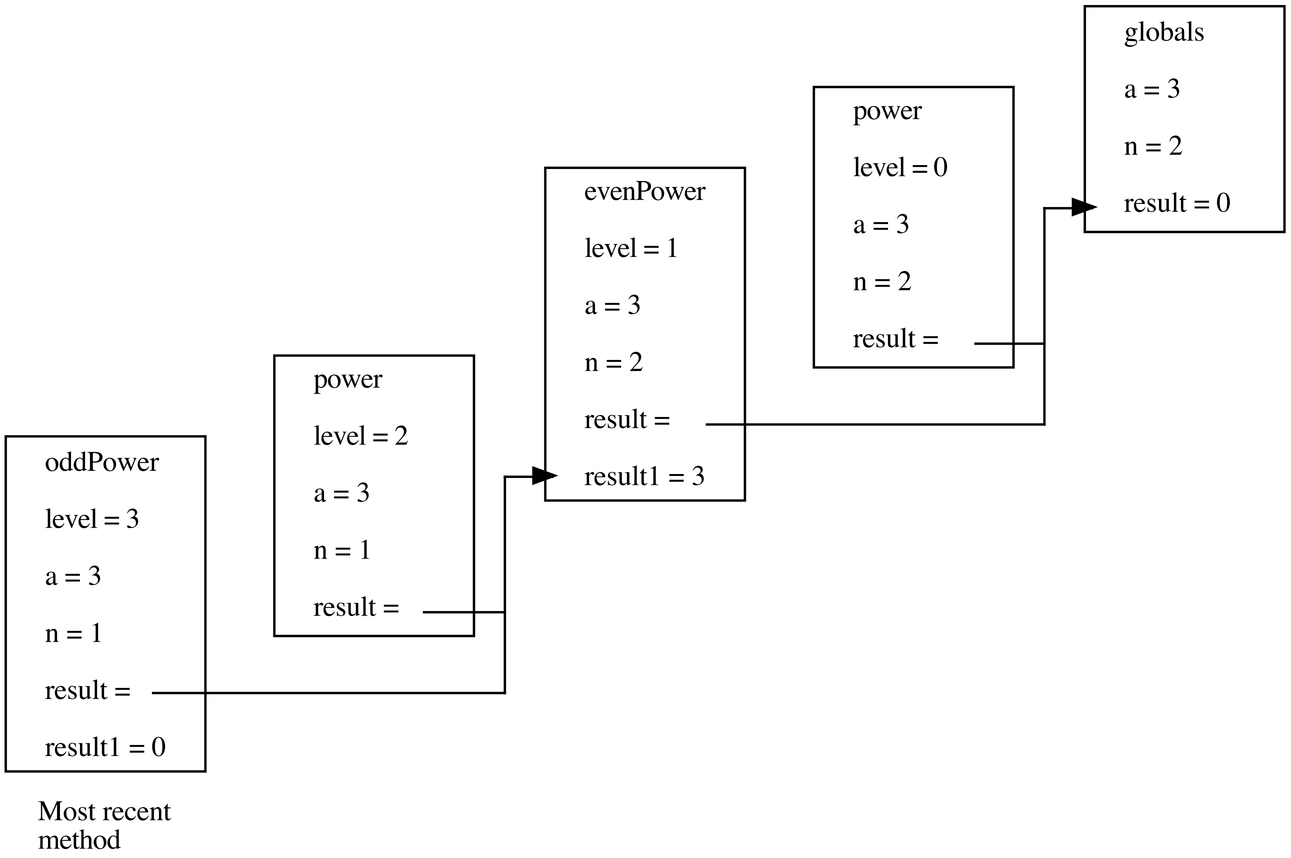
```
----------------------------------------------------------------
Programs/array/program.in
----------------------------------------------------------------
int i;
[ 10 ]int a;
int b;
for i = 0; i < 10; i++ do
     a[ i ] = i * i;
end
for i = 0; i < 10; i++ do
     printf( "%d squared = %d\n", i, a[ i ] );
end
```

```
----------------------------------------------------------------
Programs/array/interp.out
----------------------------------------------------------------
0 squared = 0
1 squared = 1
2 squared = 4
3 squared = 9
4 squared = 16
5 squared = 25
6 squared = 36
7 squared = 49
8 squared = 64
9 squared = 81
```

```
----------------------------------------------------------------
Programs/power/program.in
----------------------------------------------------------------
void evenPower( int level, a, n; ^int result; )
     begin
          int result1;
          if n == 0 then
               result^ = 1;
          else
               power( level + 1, a, n / 2, &result1 );
               result^ = result1 * result1;
          end
     end

void oddPower( int level, a, n; ^int result; )
     begin
          int result1;
          if n == 1 then
               result^ = a;
          else
               power( level + 1, a, n / 2, &result1 );
               result^ = a * result1 * result1;
          end
     end

void power( int level, a, n; ^int result; )
     begin
          if n % 2 == 0 then
               evenPower( level + 1, a, n, result );
          else
               oddPower( level + 1, a, n, result );
          end
     end
```

```
int a = 3;
int n = 2;
int result;
power( 0, a, n, &result );
printf( "%d\n", result );

---------------------------------------------------------------
Programs/power/interp.out
---------------------------------------------------------------
9
```

Consider the first invocation of “power(0, a, n, result)” in the program “power”. At the maximum level of recursion, the following run-time blocks are generated.



```
-----------------------------------------------------------------
Programs/class2/program.in
-----------------------------------------------------------------
class A
     begin
          int x, y;
          void f( int p; )
               begin
                    print( "A.f()\n" );
                    x = p;
               end
          void g( int q; )
               begin
                    print( "A.g()\n" );
                    y = q;
               end
          void printA()
               begin
                    printf( "x = %d, y = %d\n", x, y );
               end
     end

class B extends A
     begin
          int z;
          void f( int p; )
               begin
                    print( "B.f()\n" );
                    x = p;
               end
          void h( int r; )
               begin
                    print( "B.h()\n" );
                    z = r;
               end
          void printA()
               begin
                    printf( "x = %d, y = %d, z = %d\n", x, y, z );
               end
     end

B b;
^A a = b;

b.f( 444 );
b.g( 555 );
b.h( 666 );

b.printA();

a.f( 111 );
a.g( 222 );

a.printA();
b.printA();
```

Programs/class2/interp.out

```
B.f()
A.g()
B.h()
x = 444, y = 555, z = 666
B.f()
A.g()
x = 111, y = 222, z = 666
x = 111, y = 222, z = 666
```

Programs/scope/program.in

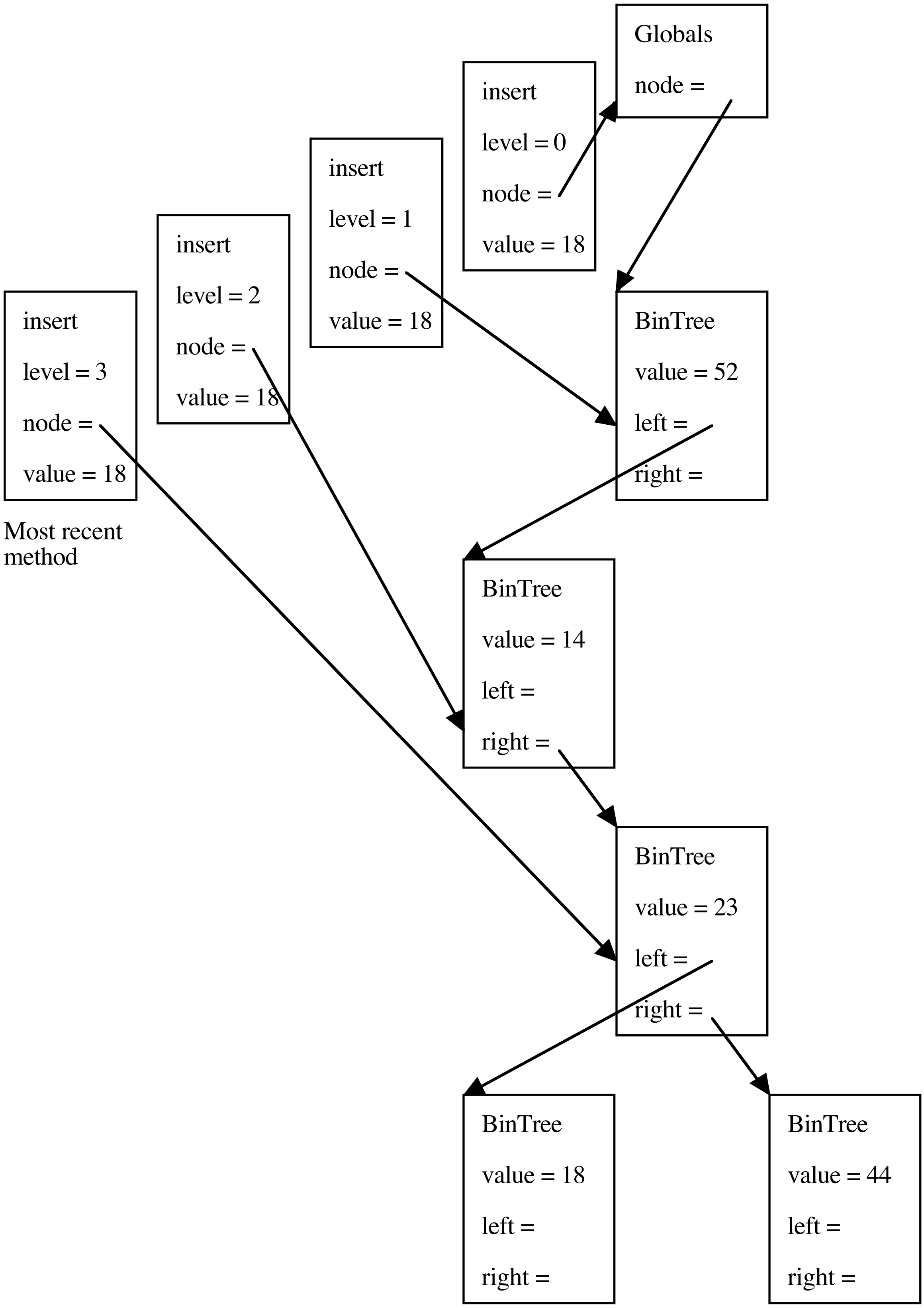
```
int z;

class A
     begin
          int w, x, y;
          [ 100 ]char result;
          void setWXY( int w, x, y; )
               begin
                    this.w = w;
                    this.x = x;
                    this.y = y;
               end
          ^char toString()
               begin
                    sprintf( result,
                         "class A: w = %d, x = %d, y = %d", w, x, y );
                    return result;
               end
     end

class B extends A
     begin
          int p, q;
          void setPQ( int p, q; )
               begin
                    this.p = p;
                    this.q = q;
               end
          ^char toString()
               begin
                    sprintf( result,
                         "class B: p = %d, q = %d, w = %d, x = %d, y = %d",
                         p, q, w, x, y );
                    return result;
               end
     end

B b;
b.setPQ( 111, 222 );
b.setWXY( 333, 444, 555 );
^A a = b;
printf( "a = %s\n", a.toString() );
```

```
----------------------------------------------------------------
Programs/scope/interp.out
----------------------------------------------------------------
a = class B: p = 111, q = 222, w = 333, x = 444, y = 555
```

```
----------------------------------------------------------------
Programs/binTree/program.in
----------------------------------------------------------------
class BinTree
     begin
          int value;
          ^BinTree left, right;
     end

^BinTree new( int level; int value; )
     begin
          ^BinTree node = nodeHeap[ freeNode++ ];
          node.value = value;
          node.left = null;
          node.right = null;
          return node;
     end

void insert( int level; ^^BinTree node; int value; )
     begin
          if node^ == null then
               node^ = new( level + 1, value );
          else
               if value < node^.value then
                    insert( level + 1, &node^.left, value );
               elif node^.value < value then
                    insert( level + 1, &node^.right, value );
               end
          end
     end

void printTree( ^BinTree node; )
     begin
          if node != null then
               printTree( node.left );
               printf( "%d ", node.value );
               printTree( node.right );
          end
     end

int freeNode = 0;
[ 10 ]BinTree nodeHeap;

^BinTree node;

insert( 0, &node, 52 );
insert( 0, &node, 14 );
insert( 0, &node, 23 );
insert( 0, &node, 44 );
insert( 0, &node, 18 ); //   Display at maximum level of recursion

printTree( node );
printf( "\n" );
```

```
--------------------------------------------------------------------
Programs/binTree/interp.out
--------------------------------------------------------------------
14 18 23 44 52
```

At the maximum level of recursion, in the invocation of "insert(0, node, 18)", the following data structures are generated. Of course, the nodes of the tree are actually part of the global block of memory.



```
----------------------------------------------------------------
Programs/exprTree/program.in
----------------------------------------------------------------
class Expr
     begin
          int kind, priority;
          ^char operator;
          ^Expr left, right;
     end

int freeNode = 0;
[ 10 ]Expr nodeHeap;

^Expr new(
     int kind, priority;
     ^char operator;
     ^Expr left, right;
     )
     begin
          ^Expr node = nodeHeap[ freeNode++ ];
          node.kind = kind;
          node.priority = priority;
          node.operator = operator;
          node.left = left;
          node.right = right;
          return node;
     end

int IDENT = 1, BIN = 2;

void parenth( int level, priority; ^Expr expr; ^char result; )
     begin
          [ 20 ]char result1;
          toString( level + 1, expr, result1 );
          if expr.priority < priority then
               sprintf( result, "(%s)", result1 );
          else
               sprintf( result, "%s", result1 );
          end
     end

void toString( int level; ^Expr expr; ^char result; )
     begin
          [ 20 ]char left, right;
          if expr.kind == IDENT then
               sprintf( result, "%s", expr.operator );
          elif expr.kind == BIN then
               parenth( level + 1, expr.priority, expr.left, left );
               parenth( level + 1, expr.priority+1, expr.right, right );
               sprintf( result, "%s%s%s", left, expr.operator, right );
          end
     end
```

```
^Expr expr1 = new( IDENT, 3, "a", null, null );
^Expr expr2 = new( IDENT, 3, "b", null, null );
^Expr expr3 = new( IDENT, 3, "c", null, null );
^Expr expr4 = new( BIN, 1, "+", expr1, expr2 );
^Expr expr = new( BIN, 2, "*", expr4, expr3 );

[ 20 ]char result;
toString( 1, expr, result );
printf( "%s\n", result );
```

Programs/exprTree/interp.out

```
(a+b)*c
```

Programs/list_concat_copy/program.in

```
class List
     begin
          int value;
          ^List next;
     end

int freeNode = 0;
[ 20 ]List nodeHeap;

^List new( int value; ^List next; )
     begin
          ^List node = nodeHeap[ freeNode++ ];
          node.value = value;
          node.next = next;
          return node;
     end

void printList( ^List a; )
     begin
          printf( "{ " );
          while a != null do
               printf( "%d", a.value );
               a = a.next;
               if a != null then
                    printf( ", " );
               end
          end
          printf( " }\n" );
     end

void concatList( ^List source1, source2; ^^List dest; )
     begin
          if source1 == null then
               dest^ = source2;
          else
               dest^ = new( source1.value, null );
               concatList( source1.next, source2, &dest^.next );
          end
     end
```

```
^List source1, source2, a2, a4, a7, a9, dest;
a9 = new( 9, null );
a7 = new( 7, a9 );
a4 = new( 4, null );
a2 = new( 2, a4 );
source1 = a7;
source2 = a2;
concatList( source1, source2, &dest );
printList( source1 );
printList( source2 );
printList( dest );
```

--
Programs/list_concat_copy/interp.out
--

```
{ 7, 9 }
{ 2, 4 }
{ 7, 9, 2, 4 }
```

--
Programs/list_delete/program.in
--

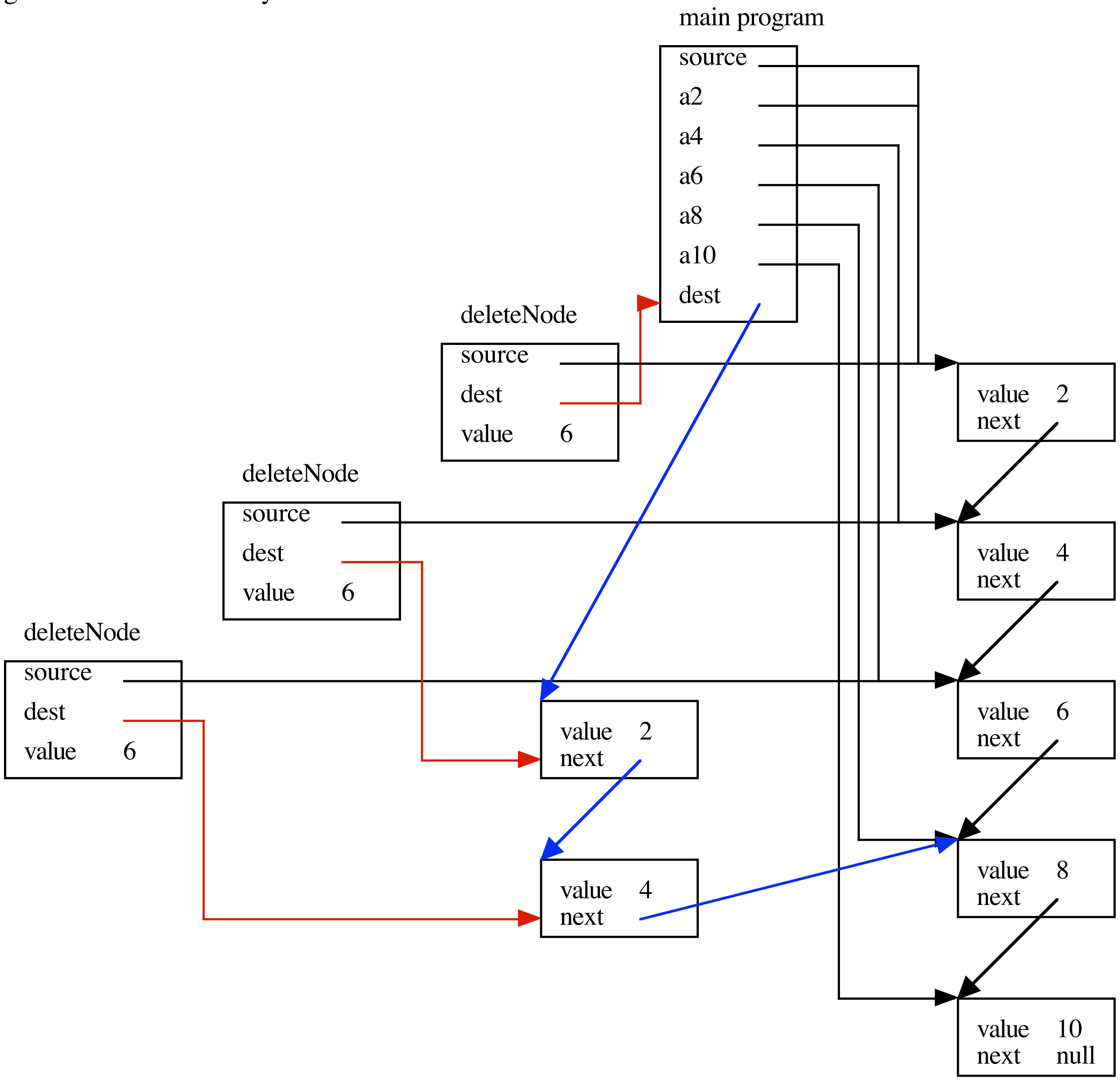
```
class List
     begin
          int value;
          ^List next;
     end

int freeNode = 0;
[ 20 ]List nodeHeap;

^List new( int value; ^List next; )
     begin
          ^List node = nodeHeap[ freeNode++ ];
          node.value = value;
          node.next = next;
          return node;
     end

void printList( ^List a; )
     begin
          printf( "{ " );
          while a != null do
               printf( "%d", a.value );
               a = a.next;
               if a != null then
                    printf( ", " );
               end
          end
          printf( " }\n" );
     end
```

```
void deleteNode( ^List source; ^^List dest; int value; )
     begin
          if source == null || value < source.value then
               dest^ = source;
          elif value == source.value then
               dest^ = source.next;
          else
               dest^ = new( source.value, null );
               deleteNode( source.next, &dest^.next, value );
          end
     end

^List source, a2, a4, a6, a8, a10, dest;
a10 = new( 10, null );
a8 = new( 8, a10 );
a6 = new( 6, a8 );
a4 = new( 4, a6 );
a2 = new( 2, a4 );
source = a2;
deleteNode( source, &dest, 6 );
printList( source );
printList( dest );
```

Programs/list_delete/interp.out

```
{ 2, 4, 6, 8, 10 }
{ 2, 4, 8, 10 }
```

At the maximum level of recursion, in the invocation of “deleteNode(source, &dest, 6)”, the following data structures are generated. Of course, the nodes of the tree are actually part of the global block of memory.



```
----------------------------------------------------------------
Programs/list_deleteAll/program.in
----------------------------------------------------------------
class List
     begin
          int value;
          ^List next;
     end

int freeNode = 0;
[ 20 ]List nodeHeap;
```

```
^List new( int value; ^List next; )
     begin
          ^List node = nodeHeap[ freeNode++ ];
          node.value = value;
          node.next = next;
          return node;
     end

void printList( ^List a; )
     begin
          printf( "{ " );
          while a != null do
               printf( "%d", a.value );
               a = a.next;
               if a != null then
                    printf( ", " );
               end
          end
          printf( " }\n" );
     end

void deleteAll( int value; ^List source; ^^List dest; )
     begin
          if source == null then
               dest^ = null;
          elif source.value == value then
               deleteAll( value, source.next, dest );
          else
               dest^ = new( source.value, null );
               deleteAll( value, source.next, &dest^.next );
          end
     end

^List source, a1, a2a, a4, a7, a2b, a9, dest;
a9 = new( 9, null );
a2b = new( 2, a9 );
a7 = new( 7, a2b );
a4 = new( 4, a7 );
a2a = new( 2, a4 );
a1 = new( 1, a2a );
source = a1;
deleteAll( 2, source, &dest );
printList( source );
printList( dest );
```

--

Programs/list_deleteAll/interp.out

--

```
{ 1, 2, 4, 7, 2, 9 }
{ 1, 4, 7, 9 }
```

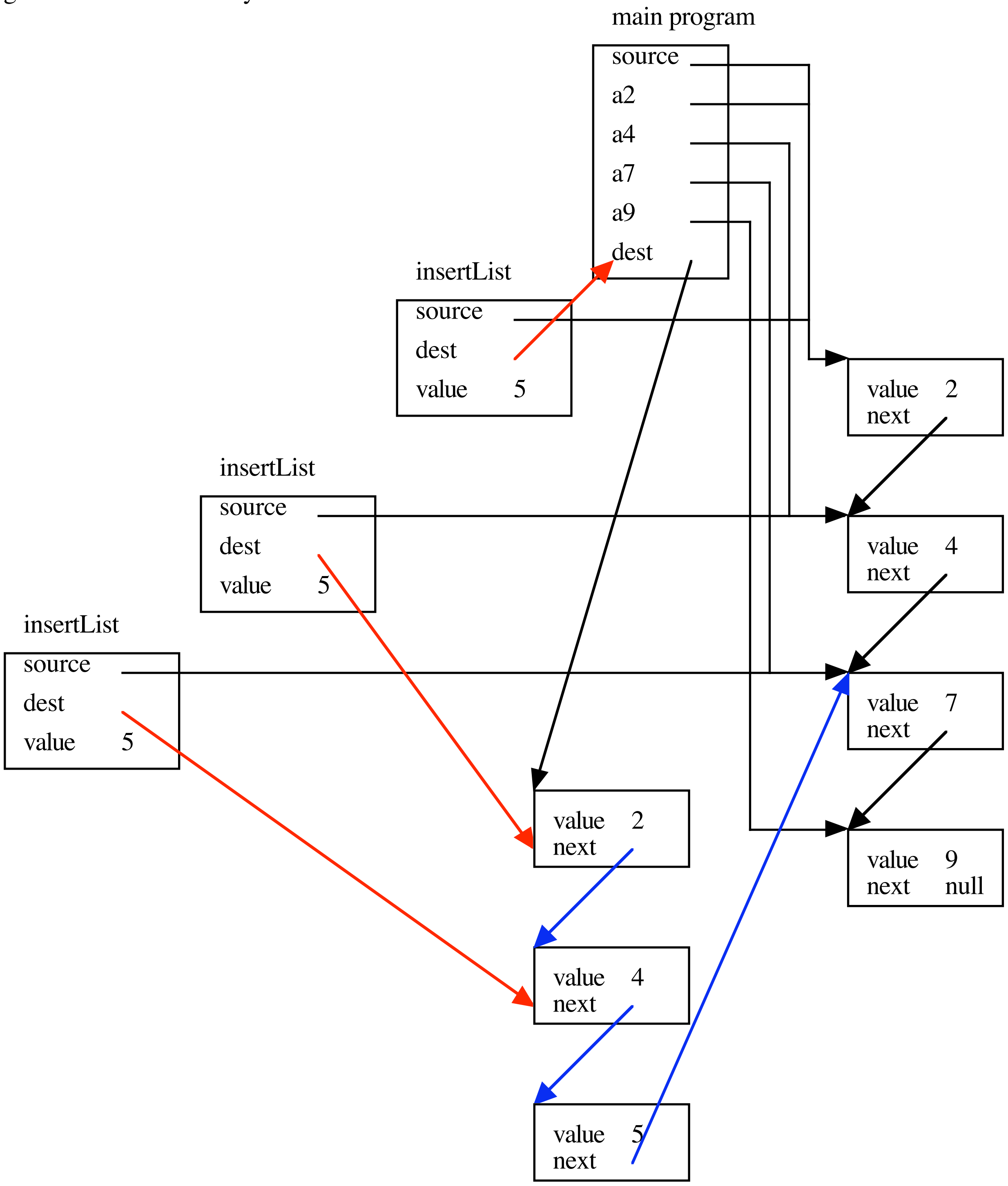
```
----------------------------------------------------------------
Programs/list_head/program.in
----------------------------------------------------------------
class List
     begin
          int value;
          ^List next;
     end

int freeNode = 0;
[ 20 ]List nodeHeap;

^List new( int value; ^List next; )
     begin
          ^List node = nodeHeap[ freeNode++ ];
          node.value = value;
          node.next = next;
          return node;
     end

void printList( ^List a; )
     begin
          printf( "{ " );
          while a != null do
               printf( "%d", a.value );
               a = a.next;
               if a != null then
                    printf( ", " );
               end
          end
          printf( " }\n" );
     end

void head( int n; ^List source; ^^List dest; )
     begin
          if n == 0 then
               dest = null;
          else
               dest^ = new( source.value, null );
               head( n - 1, source.next, &dest^.next );
          end
     end

^List source, a2, a4, a7, a9, dest;
a9 = new( 9, null );
a7 = new( 7, a9 );
a4 = new( 4, a7 );
a2 = new( 2, a4 );
source = a2;
head( 2, source, &dest );
printList( source );
printList( dest );
```

```
----------------------------------------------------------------
Programs/list_head/interp.out
----------------------------------------------------------------
{ 2, 4, 7, 9 }
{ 2, 4 }
```

```
-----------------------------------------------------------------
Programs/list_insert_copy/program.in
-----------------------------------------------------------------
class List
     begin
          int value;
          ^List next;
     end

int freeNode = 0;
[ 20 ]List nodeHeap;

^List new( int value; ^List next; )
     begin
          ^List node = nodeHeap[ freeNode++ ];
          node.value = value;
          node.next = next;
          return node;
     end

void printList( ^List a; )
     begin
          printf( "{ " );
          while a != null do
               printf( "%d", a.value );
               a = a.next;
               if a != null then
                    printf( ", " );
               end
          end
          printf( " }\n" );
     end

void insertList( ^List source; ^^List dest; int value; )
begin
     if source == null || value <= source.value then
          dest^ = new( value, source );
     else
          dest^ = new( source.value, null );
          insertList( source.next, &dest^.next, value );
     end
end

^List source, a2, a4, a7, a9, dest;
a9 = new( 9, null );
a7 = new( 7, a9 );
a4 = new( 4, a7 );
a2 = new( 2, a4 );
source = a2;
insertList( source, &dest, 5 );
printList( source );
printList( dest );
```

```
-----------------------------------------------------------------
Programs/list_insert_copy/interp.out
-----------------------------------------------------------------
{ 2, 4, 7, 9 }
{ 2, 4, 5, 7, 9 }
```

At the maximum level of recursion, in the invocation of “insertList(source, &dest, 5)”, the following data structures are generated. Of course, the nodes of the tree are actually part of the global block of memory.



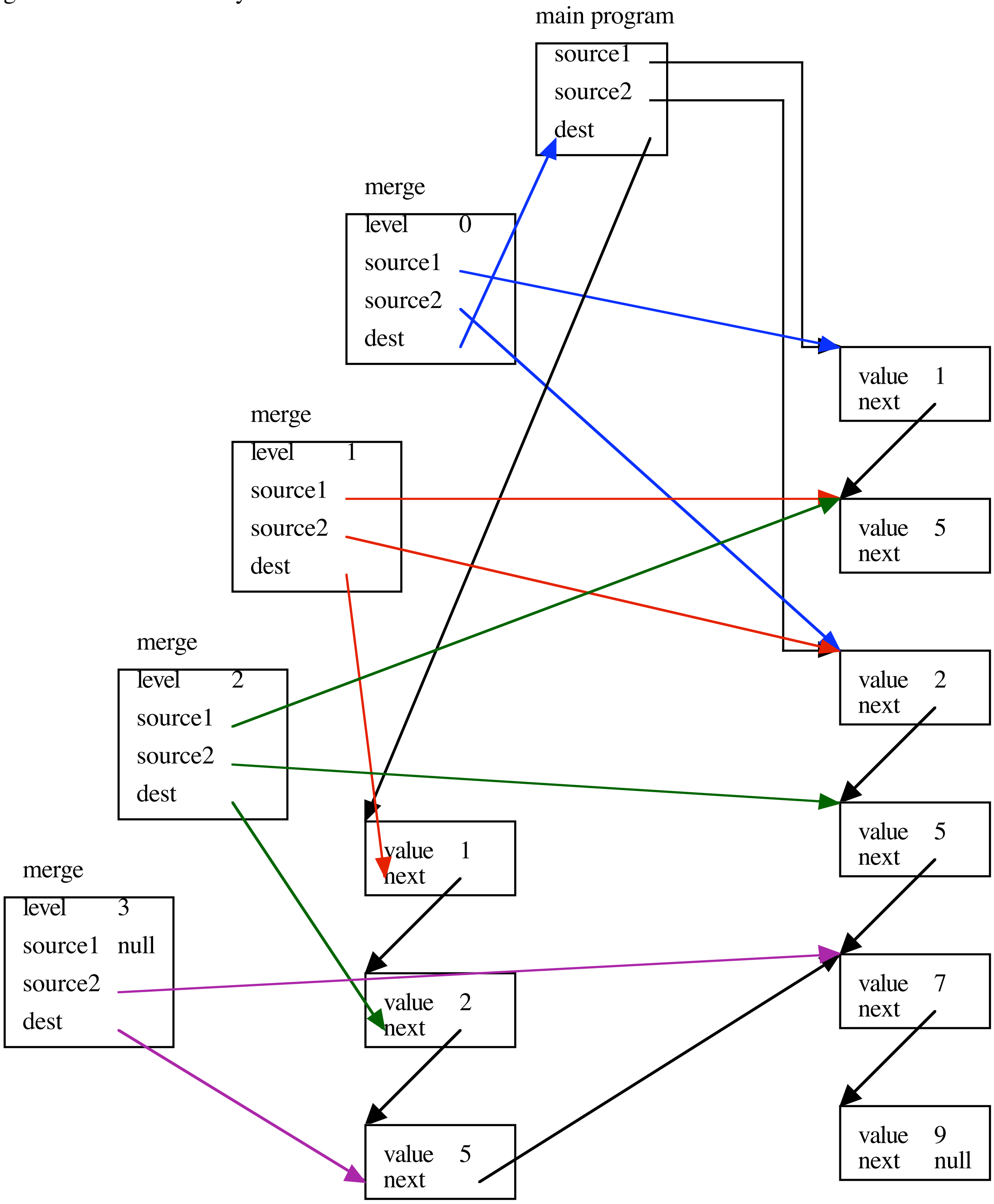
```
-------------------------------------------------------------------
Programs/list_merge/program.in
-------------------------------------------------------------------
class List
     begin
          int value;
          ^List next;
     end

int freeNode = 0;
```

```
[ 20 ]List nodeHeap;

^List new( int value; ^List next; )
     begin
          ^List node = nodeHeap[ freeNode++ ];
          node.value = value;
          node.next = next;
          return node;
     end

void printList( ^List a; )
     begin
          printf( "{ " );
          while a != null do
               printf( "%d", a.value );
               a = a.next;
               if a != null then
                    printf( ", " );
               end
          end
          printf( " }" );
     end

void printlnList( ^List a; )
     begin
          printList( a );
          printf( "\n" );
     end

void displayInfo( ^char text; int level; ^List source1, source2, dest; )
     begin
          int i;
          for i = 0; i < level; i++ do
               print( "    " );
          end
          printf( "%s( %d, ", text, level );
          printList( source1 );
          printf( ", " );
          printList( source2 );
          printf( ", " );
          printList( dest );
          printf( " )\n" );
     end

void merge( int level; ^List source1, source2; ^^List dest; )
     begin
          displayInfo( "Enter merge", level, source1, source2, dest^ );
          if source1 == null then
               dest^ = source2;
          elif source2 == null then
               dest^ = source1;
          elif source1.value < source2.value then
               dest^ = new( source1.value, null );
               merge( level + 1, source1.next, source2, &dest^.next );
          elif source1.value > source2.value then
               dest^ = new( source2.value, null );
               merge( level + 1, source1, source2.next, &dest^.next );
          elif source1.value == source2.value then
               dest^ = new( source1.value, null );
               merge( level + 1, source1.next, source2.next, &dest^.next );
          end
          displayInfo( "Exit merge", level, source1, source2, dest^ );
     end

^List source1 =
     new( 1,
     new( 5,
          null ) );
^List source2 =
     new( 2,
     new( 5,
     new( 7,
     new( 9,
          null ) ) ) );
^List dest = null;
merge( 0, source1, source2, &dest );  // Inside this invocation
printlnList( source1 );
printlnList( source2 );
printlnList( dest^ );
```

Programs/list_merge/interp.out

```
Enter merge( 0, { 1, 5 }, { 2, 5, 7, 9 }, {  } )
    Enter merge( 1, { 5 }, { 2, 5, 7, 9 }, {  } )
        Enter merge( 2, { 5 }, { 5, 7, 9 }, {  } )
            Enter merge( 3, {  }, { 7, 9 }, {  } )
            Exit merge( 3, {  }, { 7, 9 }, { 7, 9 } )
        Exit merge( 2, { 5 }, { 5, 7, 9 }, { 5, 7, 9 } )
    Exit merge( 1, { 5 }, { 2, 5, 7, 9 }, { 2, 5, 7, 9 } )
Exit merge( 0, { 1, 5 }, { 2, 5, 7, 9 }, { 1, 2, 5, 7, 9 } )
{ 1, 5 }
{ 2, 5, 7, 9 }
{ 1, 2, 5, 7, 9 }
```

At the maximum level of recursion, in the invocation of "merge(0, source1, source2, &dest)", the following data structures are generated. Of course, the nodes of the tree are actually part of the global block of memory.



--
Programs/list_reverse/program.in
--

```
class List
     begin
          int value;
          ^List next;
     end

int freeNode = 0;
[ 20 ]List nodeHeap;

^List new( int value; ^List next; )
     begin
          ^List node = nodeHeap[ freeNode++ ];
          node.value = value;
          node.next = next;
          return node;
     end

void printList( ^List a; )
     begin
          printf( "{ " );
          while a != null do
               printf( "%d", a.value );
               a = a.next;
               if a != null then
                    printf( ", " );
               end
          end
          printf( " }\n" );
     end

void reverseTransferList( ^List source; ^^List dest; )
     begin
          if source != null then
               dest^ = new( source.value, dest^ );
               reverseTransferList( source.next, dest );
          end
     end

^List source, a2, a4, a7, a9, dest;
a9 = new( 9, null );
a7 = new( 7, a9 );
a4 = new( 4, a7 );
a2 = new( 2, a4 );
source = a2;
reverseTransferList( source, &dest );
printList( source );
printList( dest );
```

--
Programs/list_reverse/interp.out
--

```
{ 2, 4, 7, 9 }
{ 9, 7, 4, 2 }
```