

Dissection of AVL tree operations

CS 202 – Fundamental Structures of
Computer Science II

Bilkent University
Computer Engineering Department

CS 202, Spring 2003

Fundamental Structures of Computer Science II
Bilkent University

1

Passing arguments to a function

- There are 3 ways in C++ to pass arguments to a function:
 - Call by value
 - Call by reference with reference arguments
 - Call by reference with pointer arguments
- Call by value
 - When an argument is passed to a function using call-by-value method, a *copy* of the argument is made and used inside the function. Changes to the copy do not affect the original caller's value in the caller.

CS 202, Spring 2003

Fundamental Structures of Computer Science II
Bilkent University

2

■ Call by reference with reference arguments

- A reference parameter is not a copy, but an alias of the original variable in the caller that is used as the arguments to a function.
- Any change that is made to the reference parameters is also reflected to the corresponding argument.
- In Call by reference with reference arguments:
 - A reference parameters is defined by putting & sign after the data type of the parameters
 - A reference argument is passed just giving the name of he variable in the caller.
 - A reference parameter is used inside a function it is defined by just giving its name.

```
int square_value (int n)
{
    n = n * n;
    return (n);
}
int square_ref (int &n)
{
    n = n * n;
    return (n);
}
void main()
{
    int x = 15;

    cout << "result = " << square_value(x) << endl;
    cout << "result = " << square_ref(x) << endl;
}
```

The diagram illustrates the flow of variables between functions. An arrow labeled "Call-by-value parameter" points from the parameter `n` in the `square_value` function to its local variable. Another arrow labeled "Reference parameter" points from the parameter `&n` in the `square_ref` function to its local variable. A third arrow labeled "argument" points from the variable `x` in the `main` function down to the call to `square_ref`.

```

int square_point (int *n)
{
    *n = (*n) * (*n);
    return (*n);
}

void main()
{
    int x = 15;

    cout << "result = " << square_pointer(&x) << endl;
}

```

The diagram illustrates the interaction between a function and its caller. A bracket labeled 'pointer parameter' points from the line 'int *n' in the function definition to the line 'cout << "result = " << square_pointer(&x)' in the main function, indicating that the address of variable x is passed to the function. Another bracket labeled 'Pointer arguments' points from the line '&x' in the main function to the same line in the function definition, indicating that the value of variable x is passed by reference.

AVL Tree Node declaration

```

template <class Comparable>
class AvlTree;

template <class Comparable>
class AvlNode
{
    Comparable element;
    AvlNode    *left;
    AvlNode    *right;
    int        height;

    AvlNode (const Comparable &theElement, AvlNode *lt,
             AvlNode *rt, int h = 0) : element (theElement),
                                         left(lt), right(rt), height (h) { }

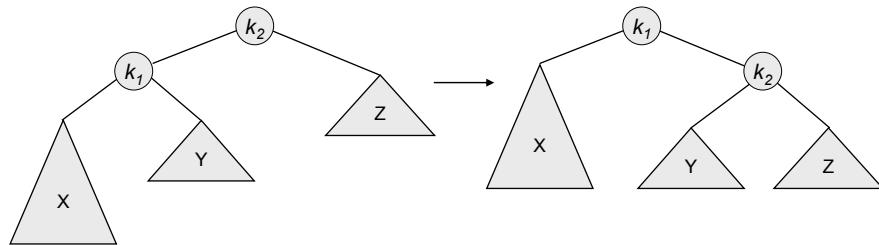
    friend class AvlTree<Comparable>;
}

template class <Comparable>
int AvlTree<Comparable>::height( AvlNode<Comparable> *t) const
{
    return t == NULL ? -1 : t->height;
}

```

AVL Tree code: single right rotation

```
1 Template <class Comparable>
2 void AvlTree <Comparable>
3 rotateWithLeftChild {AvlNode <Comparable> * &k2) const
4 {
5     AvlNode <Comparable> *k1 = k2->left;
6     k2->left = k1->right;
7     k1->right = k2;
8     k2->height = max (height (k2->left), height (k2->right)) + 1;
9     k1->height = max (height (k1->left), k2->height) + 1;
10    k2 = k1;
11 }
```



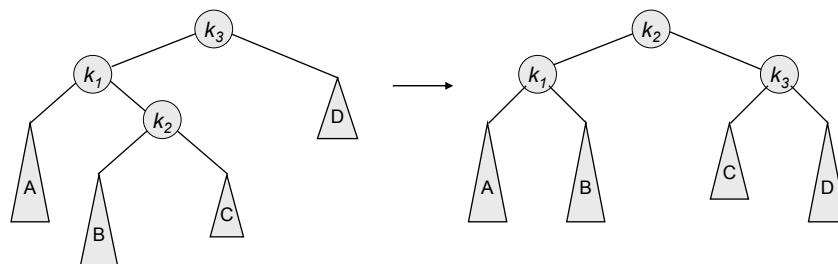
CS 202, Spring 2003

Fundamental Structures of Computer Science II
Bilkent University

7

AVL Tree code: double rotation

```
1 Template <class Comparable>
2 void AvlTree <Comparable>
3 doubleWithLeftChild {AvlNode <Comparable> * &k3) const
4 {
5     rotateWithRightChild (k3->left);
6     rotateWithLeftChild (k3);
7 }
```



CS 202, Spring 2003

Fundamental Structures of Computer Science II
Bilkent University

8

```

1 Template <class Comparable>
2 void AvlNode <Comparable>::insert( const Comparable &x, AvlNode * &t) const
3 {
4     if (t == NULL)
5         t = new AvlNode<Comparable>( x, NULL, NULL) ;
6     else if ( x < t->element )
7     {
8         insert(x, t->left);
9         if (height(t->left) - height(t->right) ==2)
10             if (x < t->left->element)
11                 rotateWithLeftChild(t);
12             else
13                 doubleWithLeftChild(t);
14         else if (t->element < x)
15         {
16             insert(x, t->right);
17             if (height(t->right) - height(t->left) ==2)
18                 if (t->right->element < x )
19                     rotateWithRightChild(t);
20                 else
21                     doubleWithRightChild(t);
22         }
23     else
24         ; // duplicate – do nothing
25
26     t->height = max(height(t->left), height(t->right))+1;
27 }

```

CS 202, Spring 2003

Fundamental Structures of Computer Science II
Bilkent University

9

```

1 int          n = 0 ;    /* an element that will be inserted */
2 AvlNode<int> * avl = NULL; /* the root of AVL tree */
3
4 void main()
5 {
6     n = 3;
7     insert(n, avl);
8
9     n = 2;
10    insert(n, avl);
11
12    n = 1;
13    insert(n, avl);
14
15    n = 4;
16    insert(n, avl);
17
18    n = 5;
19    insert(n, avl);
}

```

CS 202, Spring 2003

Fundamental Structures of Computer Science II
Bilkent University

10