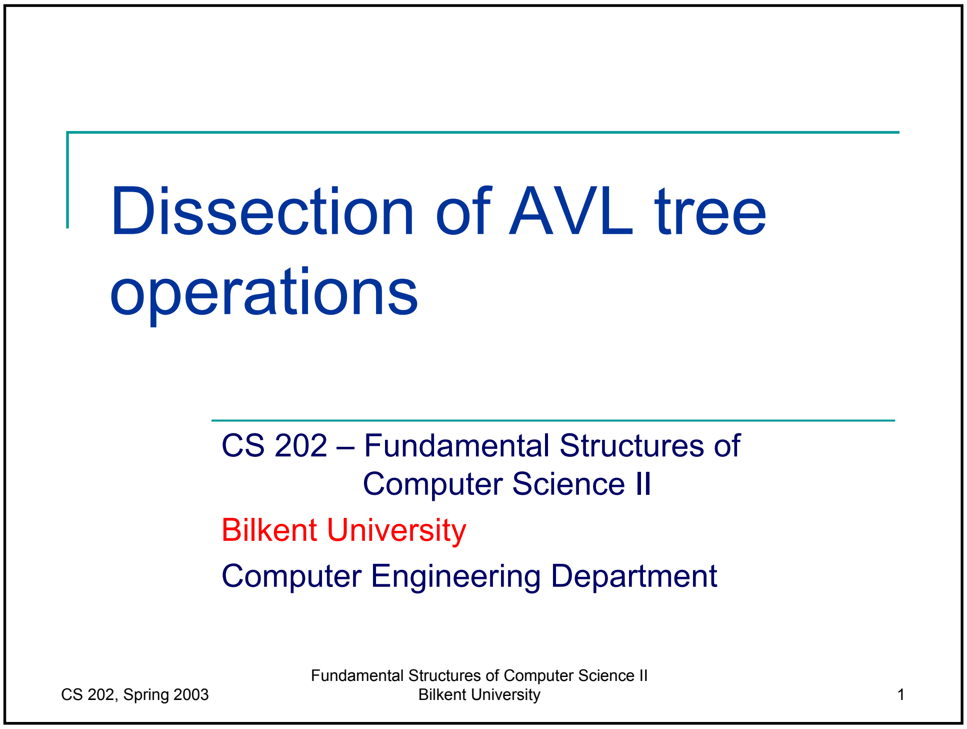
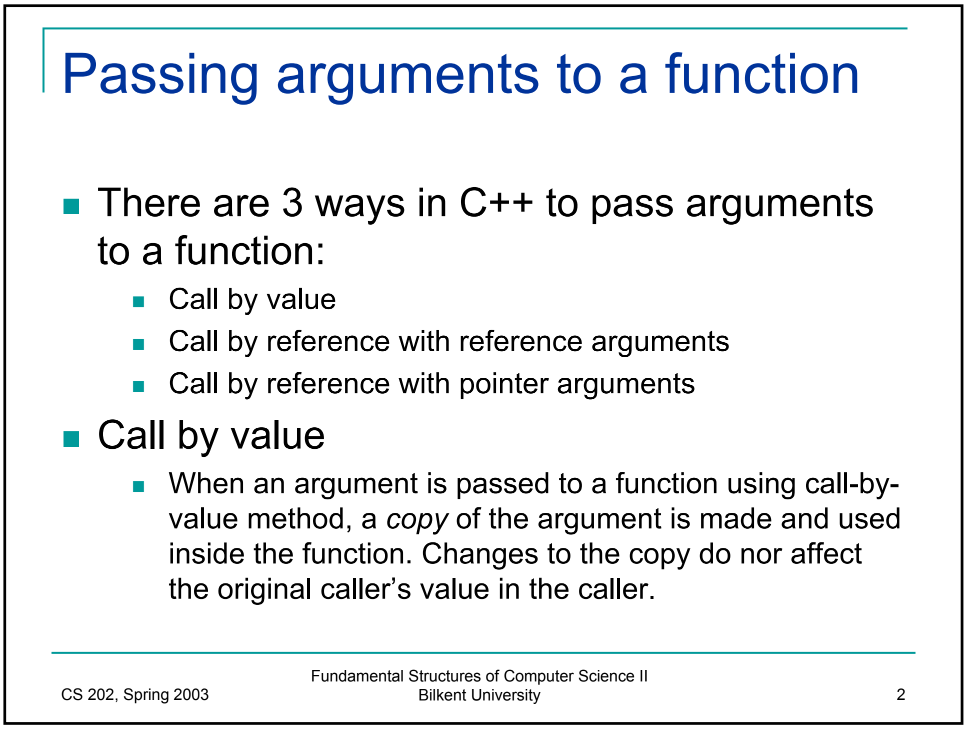


Dissection of AVL tree operations

CS 202 – Fundamental Structures of Computer Science II

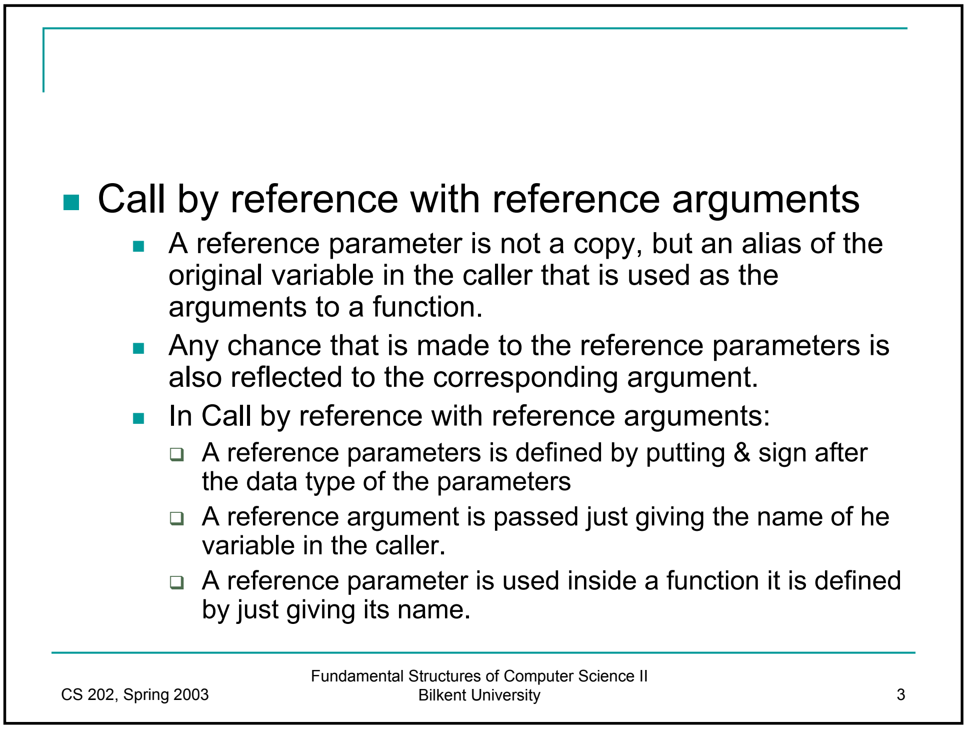
Bilkent University

Computer Engineering Department

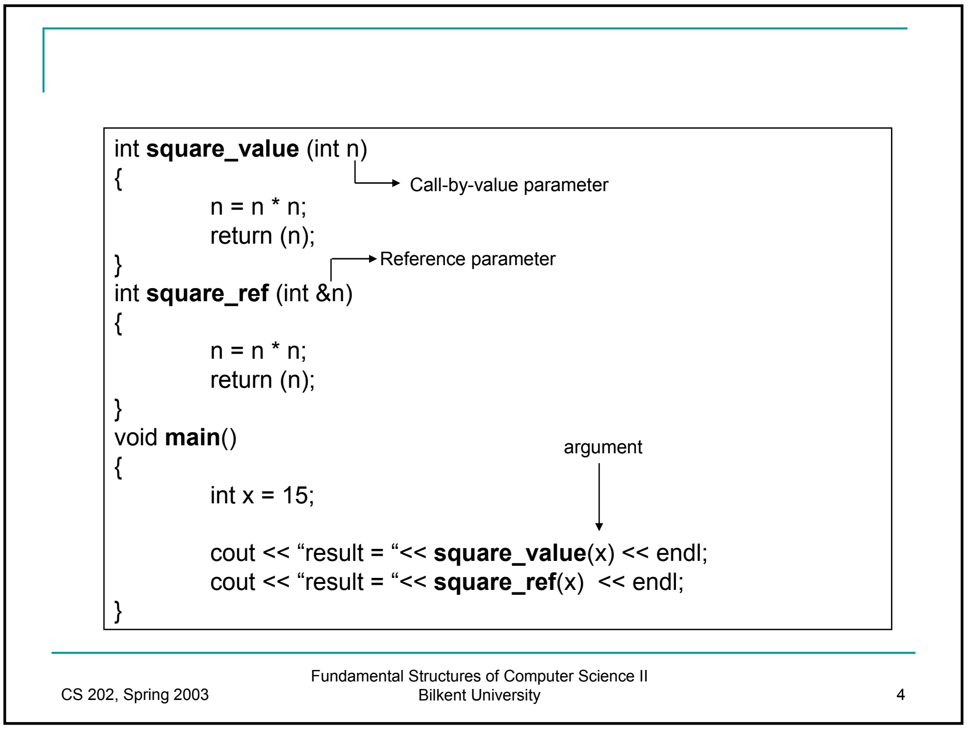


Passing arguments to a function

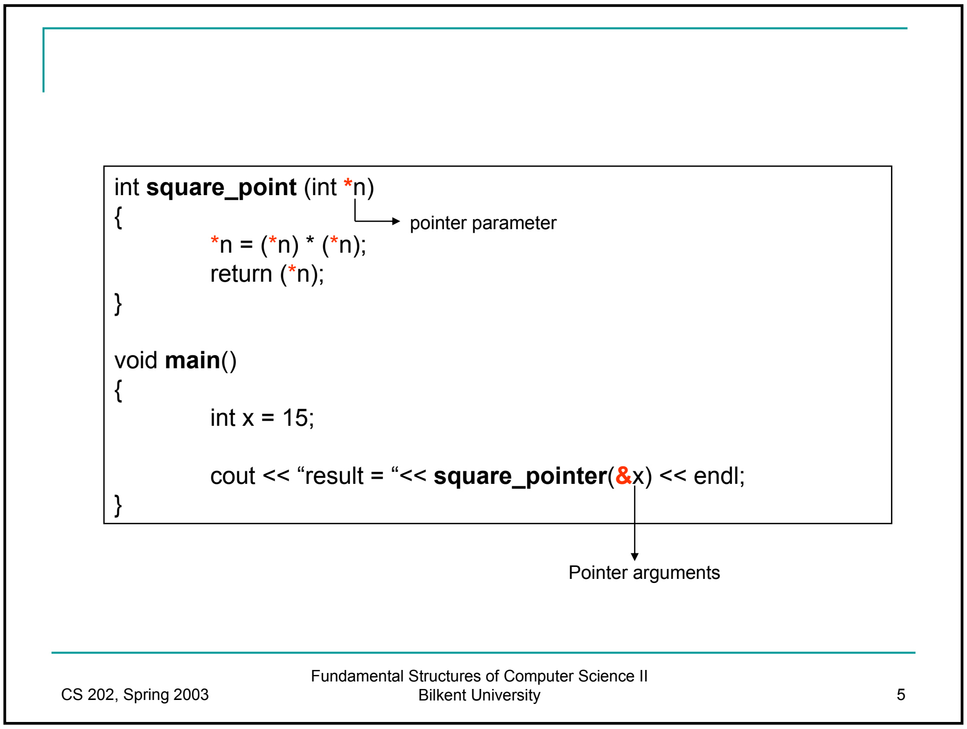
- There are 3 ways in C++ to pass arguments to a function:
  - Call by value
  - Call by reference with reference arguments
  - Call by reference with pointer arguments
- Call by value
  - When an argument is passed to a function using call-by-value method, a *copy* of the argument is made and used inside the function. Changes to the copy do nor affect the original caller’s value in the caller.



- 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 chance 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.



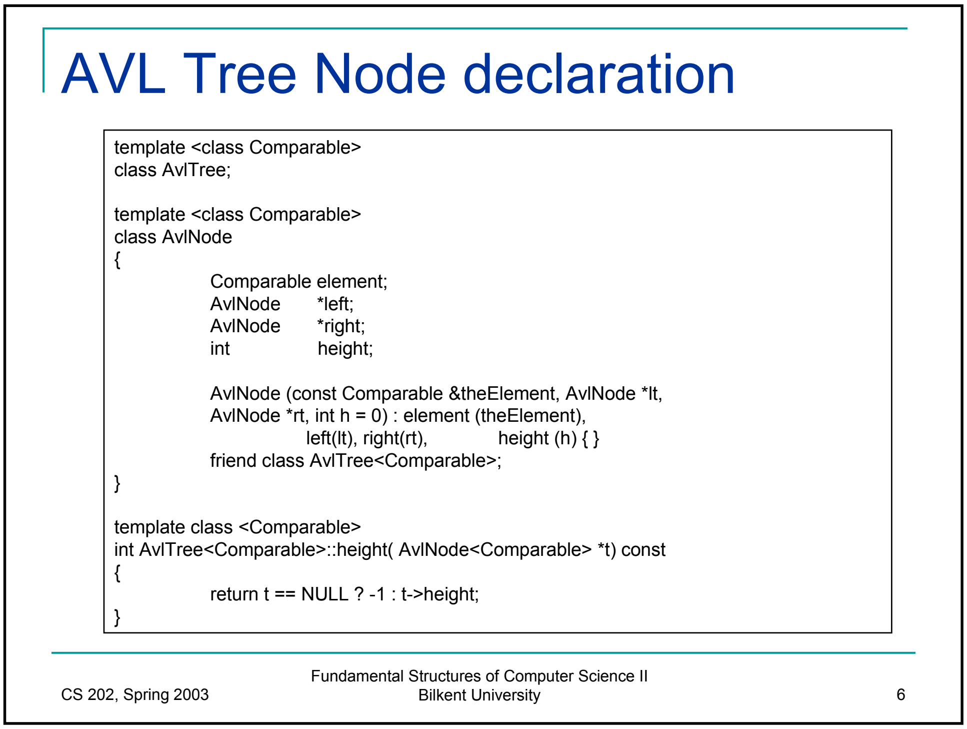
```
int square_value (int n)
{                        └──> Call-by-value parameter
        n = n * n;
        return (n);
}                     ┌──> Reference parameter
int square_ref (int &n)
{
        n = n * n;
        return (n);
}
void main()                              argument
{                                           │
        int x = 15;                         ▼

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



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

void main()
{
            int x = 15;

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

pointer parameter

Pointer arguments

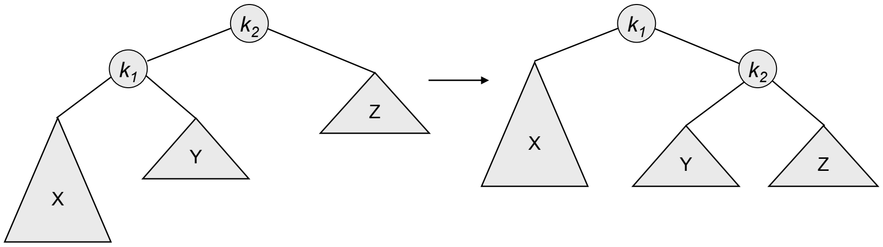


AVL Tree Node declaration

```
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;
}
```

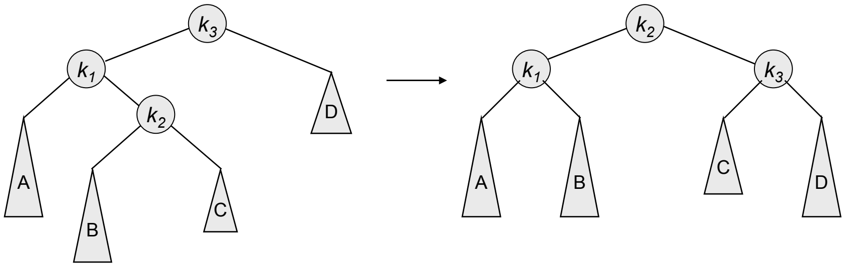
AVL Tree code: single right rotation

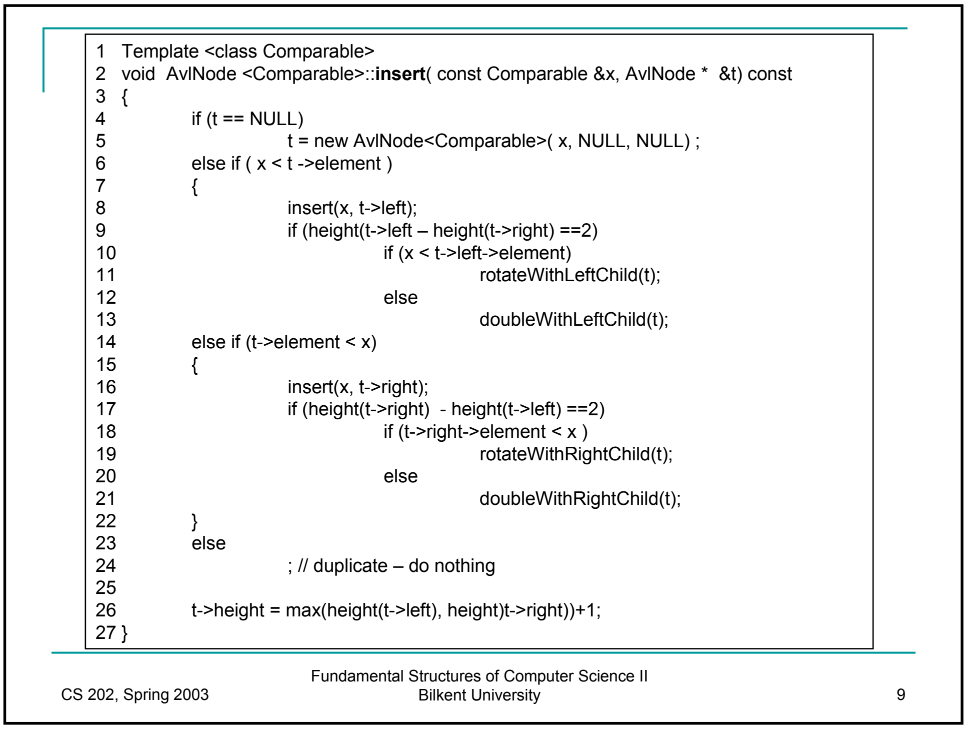
```
Template <class Comparable>
void AvlTree <Comparable>
rotateWithLeftChild {AvlNode <Comparable> *  &k2 ) const
{
            AvlNode <Comparable> *k1 = k2->left;
            k2->left = k1->right;
            k1->right = k2;
            k2->height = max (height (k2->left ), height ( k2->right) ) + 1;
            k1->height = max (height (k1->left), k2->height) + 1;
            k2 = k1;
}
```



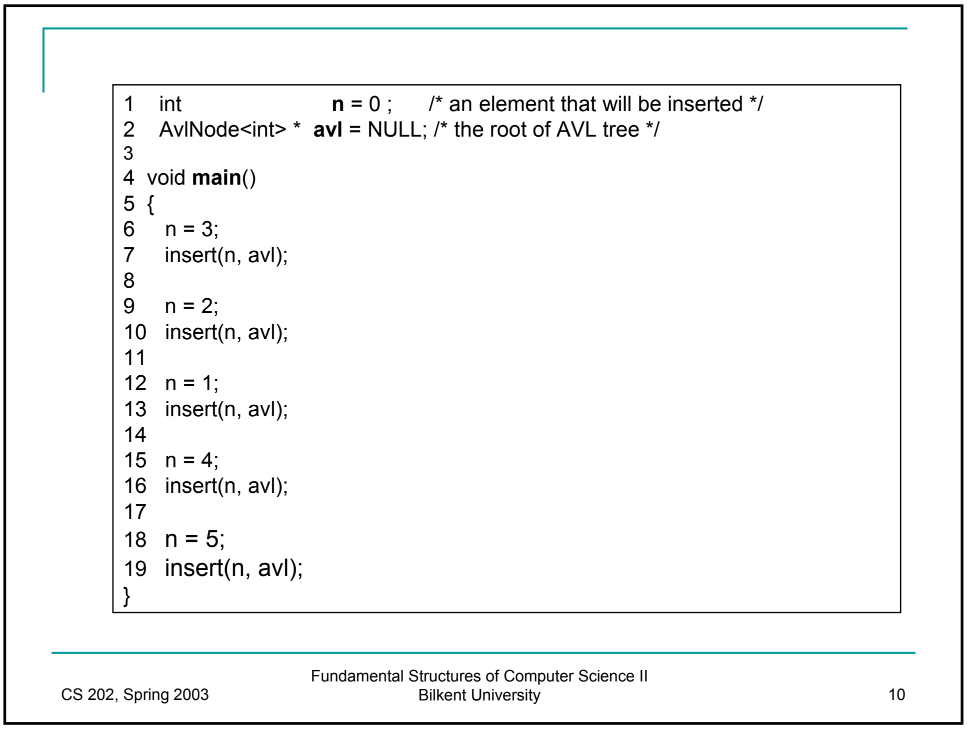
AVL Tree code: double rotation

```
Template <class Comparable>
void AvlTree <Comparable>
doubleWithLeftChild {AvlNode <Comparable> *  &k3 ) const
{
                    rotateWithRightChild (k3->left);
                    rotateWithLeftChild (k3);
}
```





```
Template <class Comparable>
void  AvlNode <Comparable>::insert( const Comparable &x, AvlNode *  &t) const
{
        if (t == NULL)
                t = new AvlNode<Comparable>( x, NULL, NULL) ;
        else if ( x < t ->element )
        {
                insert(x, t->left);
                if (height(t->left – height(t->right) ==2)
                        if (x < t->left->element)
                                rotateWithLeftChild(t);
                        else
                                doubleWithLeftChild(t);
        else if (t->element < x)
        {
                insert(x, t->right);
                if (height(t->right)  - height(t->left) ==2)
                        if (t->right->element < x )
                                rotateWithRightChild(t);
                        else
                                doubleWithRightChild(t);
        }
        else
                ; // duplicate – do nothing

        t->height = max(height(t->left), height)t->right))+1;
}
```



```
int                    n = 0 ;      /* an element that will be inserted */
AvlNode<int> *  avl = NULL; /* the root of AVL tree */

void main()
{
   n = 3;
   insert(n, avl);

   n = 2;
   insert(n, avl);

   n = 1;
   insert(n, avl);

   n = 4;
   insert(n, avl);

   n = 5;
   insert(n, avl);
}
```