

Expression Trees and the Heap

Binary Expression Trees

evaluating expressions
splitting strings in operators and operands

C++ Binary Tree of Strings

header files
defining the methods

the Heap

a heap of integer numbers
our class Heap with STL vector

- 1 Binary Expression Trees
evaluating expressions
splitting strings in operators and operands
- 2 C++ Binary Tree of Strings
header files
defining the methods
- 3 the Heap
a heap of integer numbers
our class Heap with STL vector

MCS 360 Lecture 25
Introduction to Data Structures
Jan Verschelde, 20 October 2010

Expression Trees and the Heap

Binary Expression Trees

evaluating expressions

splitting strings in operators and operands

C++ Binary Tree of Strings

header files

defining the methods

the Heap

a heap of integer numbers

our class Heap with STL vector

1 Binary Expression Trees evaluating expressions

splitting strings in operators and operands

2 C++ Binary Tree of Strings header files defining the methods

3 the Heap a heap of integer numbers our class Heap with STL vector

Binary Expression Trees

Binary Expression Trees

evaluating expressions

splitting strings in operators and operands

C++ Binary Tree of Strings

header files

defining the methods

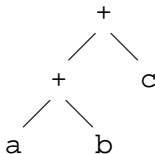
the Heap

a heap of integer numbers

our class Heap with STL vector

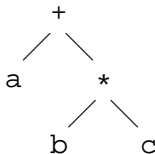
Expression trees store the evaluation order:

$a+b+c$



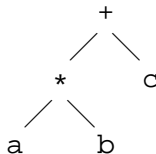
$= (a+b)+c$

$a+b*c$



$= a+(b*c)$

$a*b+c$



$= (a*b)+c$

As the children store the operands,
we first evaluate the expressions at the children.

running the program I

Binary
Expression
Treesevaluating
expressionssplitting strings in
operators and
operandsC++ Binary
Tree of Stringsheader files
defining the methods

the Heap

a heap of integer
numbers
our class Heap with
STL vector

```
$ /tmp/strtree
```

```
Give expression : a+b+c
```

```
your expression : a+b+c
```

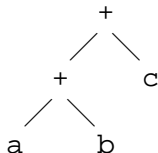
```
the expression tree :
```

```

+
  +
    a
    b
  c
postfix : ab+c+
$

```

a+b+c



= (a+b)+c

running the program II

Binary
Expression
Treesevaluating
expressionssplitting strings in
operators and
operandsC++ Binary
Tree of Stringsheader files
defining the methods

the Heap

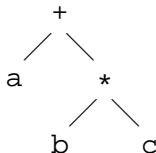
a heap of integer
numbers
our class Heap with
STL vector

```

$ /tmp/strtree
Give expression : a+b*c
your expression : a+b*c
the expression tree :
+
  a
  *
    b
    c
postfix : abc**
$

```

a+b*c



= a+(b*c)

running the program III

Binary
Expression
Treesevaluating
expressionssplitting strings in
operators and
operandsC++ Binary
Tree of Stringsheader files
defining the methods

the Heap

a heap of integer
numbers
our class Heap with
STL vector

```
$ /tmp/strtree
```

```
Give expression : a*b+c
```

```
your expression : a*b+c
```

```
the expression tree :
```

```
+
```

```
  *
```

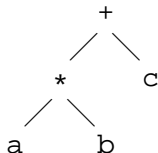
```
    a
```

```
    b
```

```
  c
```

```
postfix : ab*c+
```

```
$
```

 $a*b+c$  $= (a*b)+c$

Expression Trees and the Heap

Binary Expression Trees

evaluating
expressions
splitting strings in
operators and
operands

C++ Binary Tree of Strings

header files
defining the methods

the Heap

a heap of integer
numbers
our class Heap with
STL vector

- 1 Binary Expression Trees**
evaluating expressions
splitting strings in operators and operands
- 2 C++ Binary Tree of Strings**
header files
defining the methods
- 3 the Heap**
a heap of integer numbers
our class Heap with STL vector

expressions as strings

Binary Expression Trees

evaluating
expressions

splitting strings in
operators and
operands

C++ Binary Tree of Strings

header files

defining the methods

the Heap

a heap of integer
numbers

our class Heap with
STL vector

We consider expressions with symbolic operands.

"a+b+c" is split into "a", "+", "b", "+", "c"

We split a string into a vector of strings.

Recall:

- `find()` method on string,
- `push_back()` method on vector.

splitting strings

```
vector<string> split ( string e, string op )
{
    vector<string> r;
    int i = 0;
    if(e.find(op,i) == string::npos)
        r.push_back(e);
    else
    {
        int k;
        do
        {
            k = e.find(op,i);
            r.push_back(e.substr(i,k-i));
            r.push_back(e.substr(k,1)); i = k+1;
        }
        while(!(e.find(op,i) == string::npos));
        r.push_back(e.substr(k+1));
    }
    return r;
}
```

splitting twice

Binary
Expression
Treesevaluating
expressionssplitting strings in
operators and
operandsC++ Binary
Tree of Strings

header files

defining the methods

the Heap

a heap of integer
numbersour class Heap with
STL vector

We can split first on "+", then on "*".

```
$ /tmp/split
give expression : a+b*c
your expression : a+b*c
after split :
a
+
b*c
after second split :
 a + ( b * c )
$
```

Instead of using a vector of strings,
the split is executed recursively with a tree.

Expression Trees and the Heap

Binary Expression Trees

evaluating
expressions
splitting strings in
operators and
operands

C++ Binary Tree of Strings

header files
defining the methods

the Heap

a heap of integer
numbers
our class Heap with
STL vector

- 1 Binary Expression Trees
evaluating expressions
splitting strings in operators and operands
- 2 C++ Binary Tree of Strings
header files
defining the methods
- 3 the Heap
a heap of integer numbers
our class Heap with STL vector

```
#ifndef TREE_NODE_H
#define TREE_NODE_H
#include <string>

struct Node
{
    std::string data; // operator or operand
    Node *left;      // pointer to left branch
    Node *right;     // pointer to right branch

    Node(const std::string& s,
          Node* left_ptr = NULL,
          Node* right_ptr = NULL) :
        data(s), left(left_ptr), right(right_ptr) {}

    virtual ~Node() {}
};
#endif
```

Binary
Expression
Trees

evaluating
expressions
splitting strings in
operators and
operands

C++ Binary
Tree of Strings

header files
defining the methods

the Heap

a heap of integer
numbers
our class Heap with
STL vector

```
#ifndef MCS360_BINARY_EXPRESSION_TREE_H
#define MCS360_BINARY_EXPRESSION_TREE_H

#include "mcs360_binary_expression_node.h"

namespace mcs360_binary_expression_tree
{
    class Tree
    {
    private:

        Node *root; // data member

        // construct tree from a node
        Tree(Node *r) : root(r) {}
    };
};
```

public constructor methods

Binary Expression Trees

evaluating
expressions
splitting strings in
operators and
operands

C++ Binary Tree of Strings

header files
defining the methods

the Heap

a heap of integer
numbers
our class Heap with
STL vector

```
public:
```

```
Tree() : root(NULL) {}
```

```
Tree(const std::string& s,  
      const Tree& left = Tree(),  
      const Tree& right = Tree() ) :  
    root(new Node(s, left.root, right.root)) {}
```

Note: a tree is a pointer to a node,
though a client of `Tree` does not see the `Node` type.

other public methods

Binary
Expression
Trees

evaluating
expressions
splitting strings in
operators and
operands

C++ Binary
Tree of Strings

header files
defining the methods

the Heap

a heap of integer
numbers
our class Heap with
STL vector

```
Tree get_left() const; // returns left child
// precondition: not is_left_null()
Tree get_right() const; // returns right child
// precondition: not is_right_null();
```

```
bool is_left_null() const;
// true if left child is null
bool is_right_null() const;
// true if right child is null
```

```
std::string get_data() const;
// returns data at node
```

```
void insert(std::string e);
void insert(std::string e, std::string op);
```

Expression Trees and the Heap

Binary Expression Trees

evaluating
expressions
splitting strings in
operators and
operands

C++ Binary Tree of Strings

header files
defining the methods

the Heap

a heap of integer
numbers
our class Heap with
STL vector

- 1 Binary Expression Trees
evaluating expressions
splitting strings in operators and operands
- 2 C++ Binary Tree of Strings
header files
defining the methods
- 3 the Heap
a heap of integer numbers
our class Heap with STL vector

Binary
Expression
Trees

evaluating
expressions
splitting strings in
operators and
operands

C++ Binary
Tree of Strings

header files
defining the methods

the Heap

a heap of integer
numbers
our class Heap with
STL vector

```
#include "mcs360_binary_expression_tree.h"

namespace mcs360_binary_expression_tree
{
    Tree Tree::get_left() const {
        return Tree(root->left);
    }
    Tree Tree::get_right() const {
        return Tree(root->right);
    }
    bool Tree::is_left_null() const {
        return (root->left == NULL);
    }
    bool Tree::is_right_null() const {
        return (root->right == NULL);
    }
    std::string Tree::get_data() const {
        return root->data;
    }
}
```

the method `insert()`Binary
Expression
Trees

evaluating
expressions
splitting strings in
operators and
operands

C++ Binary
Tree of Strings

header files
defining the methods

the Heap

a heap of integer
numbers
our class Heap with
STL vector

```

void Tree::insert(std::string e)
{
    this->insert(e, "+");
}

void Tree::insert(std::string e, std::string op)
{
    using std::string;

    if(e.rfind(op) == string::npos)
    {
        if(op == "*" )
            root = new Node(e);
        else
        {
            Tree S;
            S.insert(e, "*");
            root = S.root;
        }
    }
}

```

insert() continued

Binary
Expression
Trees

evaluating
expressions
splitting strings in
operators and
operands

C++ Binary
Tree of Strings

header files
defining the methods

the Heap

a heap of integer
numbers
our class Heap with
STL vector

```
else
{
    int k = e.rfind(op);
    Tree L;
    L.insert(e.substr(0,k));
    Tree R;
    R.insert(e.substr(k+1));
    root = new Node(e.substr(k,1),
                    L.root,R.root);
}
}
```

Expression Trees and the Heap

- 1 Binary Expression Trees
evaluating expressions
splitting strings in operators and operands
- 2 C++ Binary Tree of Strings
header files
defining the methods
- 3 the Heap
a heap of integer numbers
our class Heap with STL vector

the Heap

A binary tree is a *heap* if

- 1 the root is the largest element; and
- 2 the subtrees are also heaps.

If the root is largest, we have a *max* heap.

If the root is smallest, we have a *min* heap.

The root is called the *top* of the heap.

The *bottom* of the heap is the rightmost element at the deepest level of the tree.

20 Oct 2010

Binary
Expression
Treesevaluating
expressions
splitting strings in
operators and
operandsC++ Binary
Tree of Stringsheader files
defining the methods

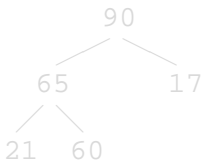
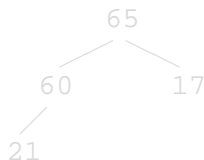
the Heap

a heap of integer
numbersour class Heap with
STL vector

storing integer numbers

pushing 21 60 17 65 90 27 70

21



20 Oct 2010

Binary
Expression
Treesevaluating
expressions
splitting strings in
operators and
operandsC++ Binary
Tree of Stringsheader files
defining the methods

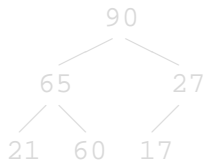
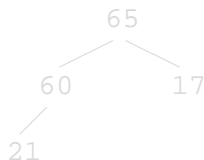
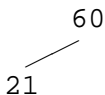
the Heap

a heap of integer
numbersour class Heap with
STL vector

storing integer numbers

pushing 21 60 17 65 90 27 70

21



20 Oct 2010

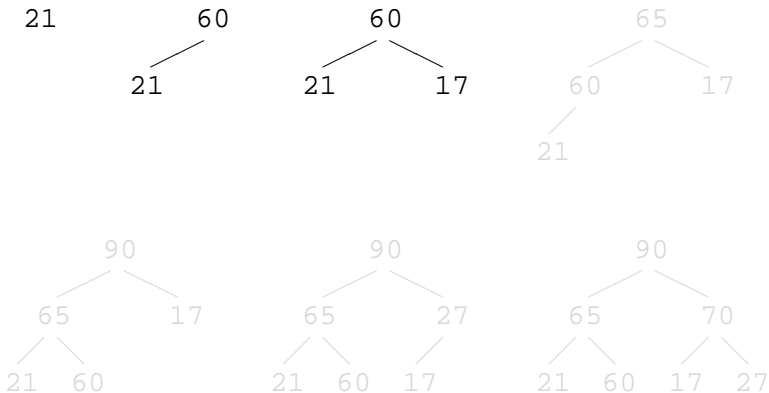
Binary
Expression
Treesevaluating
expressions
splitting strings in
operators and
operandsC++ Binary
Tree of Stringsheader files
defining the methods

the Heap

a heap of integer
numbersour class Heap with
STL vector

storing integer numbers

pushing 21 60 17 65 90 27 70



20 Oct 2010

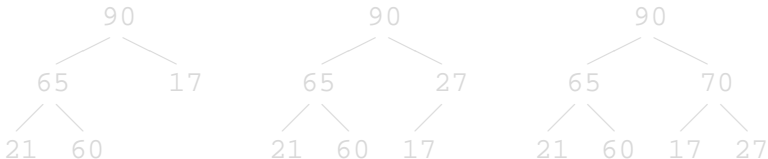
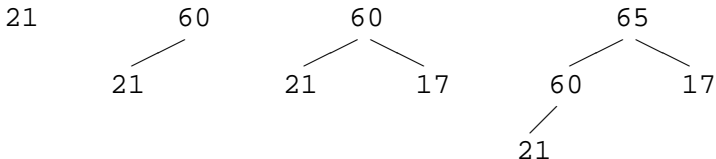
Binary
Expression
Treesevaluating
expressions
splitting strings in
operators and
operandsC++ Binary
Tree of Stringsheader files
defining the methods

the Heap

a heap of integer
numbersour class Heap with
STL vector

storing integer numbers

pushing 21 60 17 65 90 27 70



20 Oct 2010

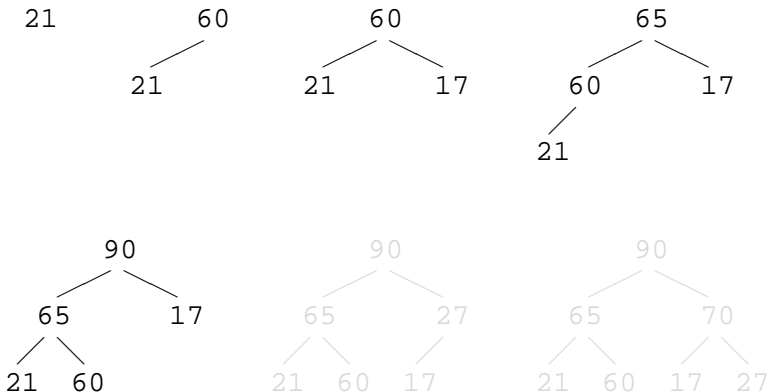
Binary
Expression
Treesevaluating
expressions
splitting strings in
operators and
operandsC++ Binary
Tree of Stringsheader files
defining the methods

the Heap

a heap of integer
numbersour class Heap with
STL vector

storing integer numbers

pushing 21 60 17 65 90 27 70



20 Oct 2010

Binary
Expression
Treesevaluating
expressions
splitting strings in
operators and
operandsC++ Binary
Tree of Stringsheader files
defining the methods

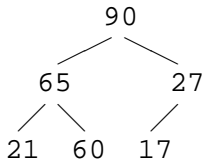
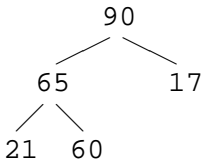
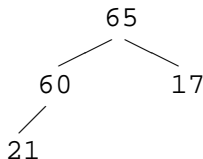
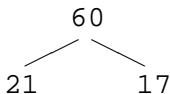
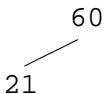
the Heap

a heap of integer
numbersour class Heap with
STL vector

storing integer numbers

pushing 21 60 17 65 90 27 70

21



20 Oct 2010

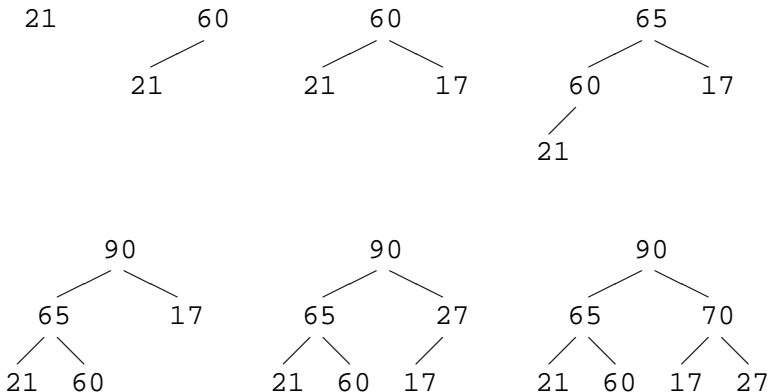
Binary
Expression
Treesevaluating
expressions
splitting strings in
operators and
operandsC++ Binary
Tree of Stringsheader files
defining the methods

the Heap

a heap of integer
numbersour class Heap with
STL vector

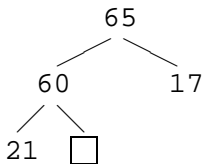
storing integer numbers

pushing 21 60 17 65 90 27 70



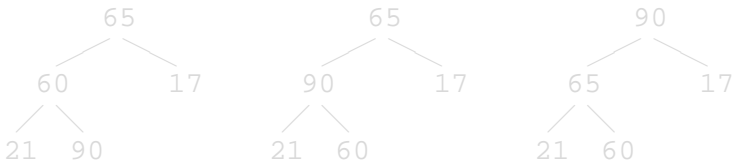
swapping numbers

Insert 90 into:



The bottom is 21, the box marks the new bottom.

As long as child is larger than parent, we swap:

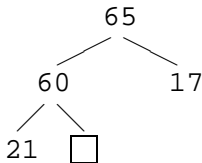


Important: #swaps is bounded by depth of the tree.

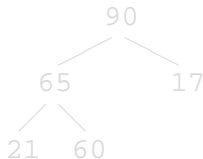
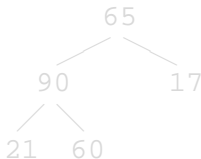
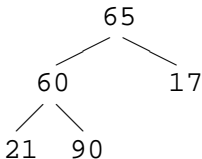
If n numbers on heap, then `push()` is $O(\log_2(n))$.

swapping numbers

Insert 90 into:



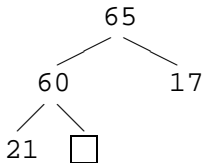
The bottom is 21, the box marks the new bottom.
As long as child is larger than parent, we swap:



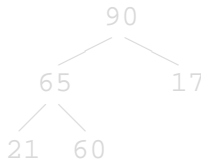
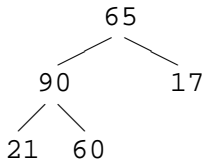
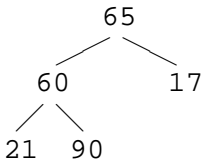
Important: #swaps is bounded by depth of the tree.
If n numbers on heap, then `push()` is $O(\log_2(n))$.

swapping numbers

Insert 90 into:



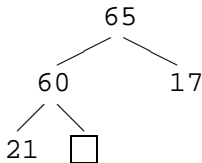
The bottom is 21, the box marks the new bottom.
As long as child is larger than parent, we swap:



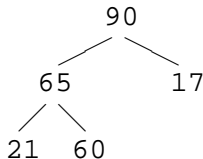
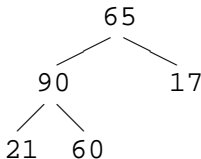
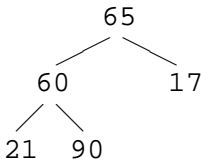
Important: #swaps is bounded by depth of the tree.
If n numbers on heap, then `push()` is $O(\log_2(n))$.

swapping numbers

Insert 90 into:



The bottom is 21, the box marks the new bottom.
As long as child is larger than parent, we swap:



Important: #swaps is bounded by depth of the tree.
If n numbers on heap, then `push()` is $O(\log_2(n))$.

storing heap as vector

Binary
Expression
Trees

evaluating
expressions
splitting strings in
operators and
operands

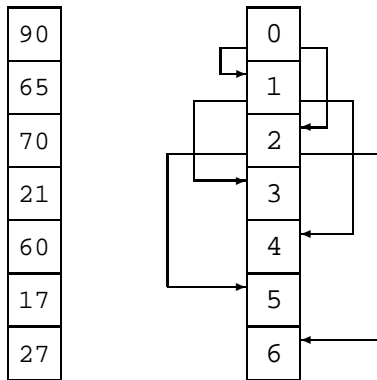
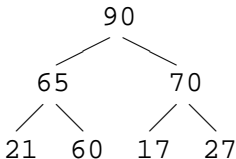
C++ Binary
Tree of Strings

header files
defining the methods

the Heap

a heap of integer
numbers

our class Heap with
STL vector



For node at p : left child is at $2p + 1$, right child is at $2p + 2$.
Parent of node at p is at $(p - 1)/2$.

Expression Trees and the Heap

- 1 Binary Expression Trees
evaluating expressions
splitting strings in operators and operands
- 2 C++ Binary Tree of Strings
header files
defining the methods
- 3 the Heap
a heap of integer numbers
our class Heap with STL vector

a class Heap

file `mcs360_integer_heap.h`

```
#ifndef MCS360_INTEGER_HEAP_H
#define MCS360_INTEGER_HEAP_H
```

```
#include<vector>
#include<string>
```

```
namespace mcs360_integer_heap
{
    class Heap
    {
        private:

            std::vector<int> h;

            int index_to_bottom;
```

Binary
Expression
Trees

evaluating
expressions
splitting strings in
operators and
operands

C++ Binary
Tree of Strings

header files
defining the methods

the Heap

a heap of integer
numbers
our class Heap with
STL vector

public:

```
Heap(); // creates empty heap
```

```
int size() const;  
// returns the size of the heap
```

```
int top() const;  
// returns the top of the heap
```

```
int bottom() const;  
// returns the bottom of the heap
```

```
void push ( int n );  
// pushes n to the heap
```

file `mcs360_integer_heap.cpp`Binary
Expression
Trees

evaluating
expressions
splitting strings in
operators and
operands

C++ Binary
Tree of Strings

header files
defining the methods

the Heap

a heap of integer
numbers
our class Heap with
STL vector

```
#include "mcs360_integer_heap.h"

namespace mcs360_integer_heap
{
    Heap::Heap() {
        index_to_bottom = -1;
    }
    int Heap::size() const {
        return index_to_bottom+1;
    }
    int Heap::top() const {
        return h[0];
    }
    int Heap::bottom() const {
        return h[index_to_bottom];
    }
}
```

the method `push()`Binary
Expression
Trees

evaluating
expressions
splitting strings in
operators and
operands

C++ Binary
Tree of Strings

header files
defining the methods

the Heap

a heap of integer
numbers
our class `Heap` with
STL vector

```
void Heap::push( int n )
{
    if(h.size() > this->size())
        h[++index_to_bottom] = n;
    else
    {
        h.push_back(n);
        index_to_bottom++;
    }
    swap_from_bottom(index_to_bottom);
}
```

swapping elements

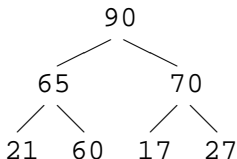
A private function member for `push()`:

```
void Heap::swap_from_bottom( int p )
{
    if(p == 0) return;

    int parent = (p-1)/2;
    if(h[parent] < h[p])
    {
        int t = h[p];
        h[p] = h[parent];
        h[parent] = t;
        swap_from_bottom(parent);
    }
}
```

converting to string

To write



the heap as vector: 90 65 70 21 60 17 27

the heap as tree:

```
90
  65
    21
    60
  70
    17
    27
```

writing to string

```
std::string Heap::to_tree_string( int k, int p )
{
    using std::ostringstream;
    ostringstream s;

    for(int i=0; i<k; i++) s << "  ";
    s << h[p] << std::endl;

    int left = 2*p+1;
    if(left > index_to_bottom) return s.str();
    s << to_tree_string(k+1,left);

    int right = 2*p+2;
    if(right > index_to_bottom) return s.str();
    s << to_tree_string(k+1,right);
    return s.str();
}
```

Binary
Expression
Trees

evaluating
expressions
splitting strings in
operators and
operands

C++ Binary
Tree of Strings

header files
defining the methods

the Heap

a heap of integer
numbers
our class Heap with
STL vector

Summary + Assignments

Binary Expression Trees

evaluating
expressions
splitting strings in
operators and
operands

C++ Binary Tree of Strings

header files
defining the methods

the Heap

a heap of integer
numbers
our class Heap with
STL vector

Introduced expression trees and started §8.5 on the heap.

Assignments:

- 1 Describe the changes needed to the binary expression trees program to deal with subtraction and division. How would you handle brackets and nesting?
- 2 Implement the changes of the previous exercise.
- 3 Generate a random sequence of 10 numbers and draw the evolution of the tree (also for every swap) when pushing the numbers onto a heap.
- 4 Define an exception `EmptyHeap` to be thrown when `top()` or `bottom()` is applied to an empty heap.

Homework due Friday 22 October, at noon:

#1 of L-15, #2 of L-16, #2,3 of L-17, #2 of L-18.