CSI 310: Lecture 19
Stack Data Structure Implementation
Expressions, Trees, and Postfix Notation
};
//this is the last.
void *operator delete(void *mem)
ValandposNode *next; //addr on next node or NULL if
Valandpos data;
};
struct ValandposNode
{

};

struct Valandpos

{ //subexpression in MYEXPR.

};

struct Valandpos

{ //int position; //index of 1st char. of that
  double value; //value of some subexpression.

};

struct Valandpos

{ //Classic C-Style Stack Implem with a Linked List.

};
{ 
    return ret;
    delete pt;
    VPStackhead = pt->next;
    VPAndpos ret = pt->data;
    VPAndpos *pt = VPStackhead;
    assert(VPStackhead);
}

VPAndpos VPpop()
{
    VPStackhead = pt;
    pt->next = VPStackhead;
    pt->data = vp;
    VPAndposNode *pt = new VPAndposNode;
}

void VPpush(VPAndpos vp) 
{
    VPAndposNode VPStackhead; //GLOBAL
    char MYEXPR[EXPRSIZE]; //GLOBAL
    #define EXPRSIZE 200

University at Albany Computer Science Dept.
MyExpr: (3 + 4) * (2 - 6)
implementation of this ValAndPos stack.

Data Structure diagram of Linked List

6 in position 10.

stacks after processing the

ValAndPos Operator ValAndPos

1)

*

-

VPstackHEAD

NULL

1

7.0

2.0

8

10

6.0

NUL

1

7.0

2.0

8

10

6.0

garbage

M_expr

199

1 2 3 4 5 6 7 8 9 10 11 12 13 14

garbage


```c
void VPPush(ValAndPos vp)
{
    VPNode *pt = new VPNode;
    pt->value = vp.value;
    pt->position = vp.position;
    //call VPPush with a Value Argument!
}
```
void VPPush(ValAndPos vp)
{
    ValAndPosNode *pt = new ValAndPosNode;
    pt->data = vp;
    pt->next = VPStackHEAD;
}

push of VPPush
([local] automatic
variable belonging
to this activation)

VPPush
VPStackHEAD
VPStackHEAD
VP StackHEAD = new ValAndPosNode;
    pt = VPStackHEAD;
    pt->data = vp;
    pt->next = VPStackHEAD;
    ValAndPosNode *pt = new ValAndPosNode;
    pt->data = vp;
    pt->next = VPStackHEAD;
    ValAndPosNode *pt = new ValAndPosNode;
    pt->data = vp;
    pt->next = VPStackHEAD;
void VPPush(ValAndPos vp) {
    ValAndPosNode *pt = VPStackHEAD;
    VPStackHEAD = pt;
    vp = ValAndPosNode *pt;
}

VPStackHEAD

0
2
3.0
VPStackHEAD = pt;

ValAndPosNode *vp

local automatic variable belonging to this activation

VPPush

2
3.0
NULL

{
{  
// you wrote:
stack<char> operations;  
stack<valandpos> valposstack;  
}

void doOneExpression(char * MYEXPR)  
...  

#include "valandpos.h"  
#include "stack2.h"  

Main and Sartch Style:  

University at Albany Computer Science Dept.
{ 
  Val*TMPstack.push(TMP);
  // Your program would generate the position
  TMP.position = 2; // Your program would compute the value
  TMP.value = 3.0; // Your program would compute the value
  Val*TMP2;
  // stack<char> operations;
  stack<Val*TMP> Val*TMPstack;
  } 

void DoOneExpression(char *MYexpr) {
  // include "ValAndPos.h"
  #include "ValAndPos.h"
  #include "stack2.h"
  
  How to push a Val*TMP record on this stack:
{
    // Print a report for debugging.
    // From the VALADDpos stack
    "\" \"position="\" TMP.position "\" count "\" We popped (value="\" TMP.value "\" VALposstack.pop();
    TMP = VALposstack.top();
    VALandpos TMP;
    ... // stack>char>operations;
    stack>VALandpos>VALposstack;
}

void DoExpression (char *MYEXPR )
#include "VALandpos.h"
#include "stack2.h"
#include "stack1.h"

How to pop and use a VALandpos record on this stack:
\[(z(1z - 2z) + z(1h - 2h) + z(1x - 2x))\text{sub}

2. How can we organize computer memory so the value of each subexpression is stored after it is computed and can be retrieved when it will be needed?

in Java, classes, files...

- Programs: HTML, XML, etc.
- Web documents, PostScript documents, code
- Examples of sequences of expressions: Calculator input: C++/Java source

1. How can we program the computer to do the arithmetic or other operations in the order expressed by the expression?

Problems with Expressions:
The particular expressions input to the computer for an expression evaluator program to evaluate are NOT KNOWN AHEAD OF TIME.
Complicated algorithms are needed to find them, anywhere in the expression.

2. The first operation as well as the top level operation can be located.

I. They require precedence rules and/or ( ) to express the order of operations.

Facts about infix expressions:
Which are strings, not trees.

They don't solve the problems (1) and (2) because the users input expressions, but what about expression trees?
the subexpression values until they are used by the recursive evaluator.

3. Solution to expression problem (2):
The stack of activation records stores

order.

Recursive evaluation easily finds and executes all the operations in the right

2. The top level operation is in the root of the tree.

1. The expression tree directly reveals the order of operations.

Remember about expression trees:
Parentheses? Prefix and Postfix notation can express every expression without any parentheses. Suppose each operator has its own fixed number of operands, the expression

Postfix: Each operator is written after its operand subexpressions.

Infix: Each operator is written between its operand subexpressions.

Prefix notation: Each operator is written before its operand subexpressions.

Expressions (as strings) very smart.

Computer scientists know two alternatives to infix notation for writing...
expression in postfix form.

are used by recursive evaluation of an expression tree, you will write the

expression (solve problem (2))

An stack of subexpression values (store by push, access and removed by pop).

Expression (solve problem (1))

Exactly matches the order of data access when the computer evaluates the

left-to-right order of atoms (constants and variables) and operators.

What is cool about Postfix:
Suppose we write down the constants and operations in the order that we use them when we evaluate this expression. First we take 6 and then add them, and then 9, and then add them, and then 6, and then 4. So, we first write: 6 + 9 + 6 4. 

Next we write 3, then divide / and then 9, and then add them. When we evaluate this expression, first we take 6 when we evaluate this in the order that we use the constants and operations.

Here's the result: The expression tree:

\[
\left( \left( 4 - 6 \right) \times \left( \frac{3}{6 + 9} \right) \right)
\]

Next, we show the stack used to evaluate this postfix expression after each step of evaluation:

The stack used to evaluate this postfix expression.
We can verify the result is 10 by calculating directly from the expression tree.

We can also verify this from the expression by using elementary school methods:

\[
( ( 6 - 4 ) \times ( 3 / ( 6 + 9 ) ) )
\]

\[
6+9=15 \\
15/3=5 \\
6-4=2 \\
5\times2=10
\]
Different activations—are different automatic variables—used for automatic variables in the activation record.

The event of "calling one function once" is an activation.

Each time a function is called, an activation record is created.
4. Now, automatic variables are in the activation record of the activation
RETURNED TO.

3. THE ACTIVATION RECORD OF A CALL ORIGINALLY CREATED THIS ACTIVATION IS
"Goes away"

2. THIS ACTIVATION'S ACTIVATION RECORD

1. The return value (if any) is saved for use by the caller.

When a function ACTIVATION executes the RETURN . . . ; statement:
{ return Solution;

{ Solution = MERGE( Anst, Anst2 );
  node = MERGE( Subproblems2 )
  node = MERGE( Subproblems1 )
   //RECURSE one or more times:
Superproblem2 = Instance; //SPLIT removed half the original list.
Superproblem1 = SPLIT( Instance );
node = Subproblems1, Subproblems2;
}
else
{ Solution = Instance; //TRIVIAL SORT OF a 1-Element list!
}
if ( Length(Instance) == 1 )
  node = MERGEsort( node * Instance );
particular activation of the (recursive) function _merge_sort_ is doing.

**VERY IMPORTANT:** AUTOMATIC variables are used to keep track of what a
But, running fact(0) calculates 0! = 0 is true and returns 1.

720, then computes 720 * 7 = 5040 and returns it.

For example, running fact(7) computes 7 * 6 * 5 * 4 * 3 * 2 * 1 = 5040, which returns

```c
{ if(n==0) return 1;
  else return fact(n-1)*n;
}
```

Example:

when the function runs, calls the same function, either directly or indirectly.

**Definition:** A function is **recursive** means the body of the function sometimes calls the same function, either directly or indirectly.

- Elegant way to write programs; performance can be improved with routine.
- Powerful problem solving technique.
- Understanding not just programming, data structures and algorithms.

Recursion:

Trees, Recursion, Expressions. Stacks are closely related.


```c
{ 
    return Solution; 

    Solution = MERGE (Anst1, Anst2); 
    done * Anst2 = MERGE (Subproblem2); 
    done * Anst1 = MERGE (Subproblem1); 
    \* RECURSE one or more times: 
    Subproblem2 = Instance; \* SLIT removed half the original list. 
    Subproblem1 = SLIT (Instance); 
    done * Subproblem1, Subproblem2; 
} 
else 
{ 
    Solution = Instance; \* TRIVIAL SORT OF a 1-Element List; 
}
```

7 ( ) Length(Instance) == 1 
```
done * MERGE-SORT (done * Instance) 
```

Divide and Conquer Pattern Applied to Sorting

University at Albany Computer Science Dept.
of the half that WAS REMOVED from the linked list.

(2) The return value is the addr. of the first node
the nodes removed.

halt

\texttt{\textbackslash post: (1) Instance=ortGen\textbackslash n value, but with approx. half
\textbackslash twin\textbackslash n list of AT LEAST TWO C-strings.
\textbackslash pre: Instance=the addr. of the first node of a
node } \texttt{SPLIT ( node } \texttt{ Instance )}

C-strings will be compared using strcmp() from <cstring>.
dnode is a linked-list node type; each node's data is the address of a C-string.
Merge sort explained using Pre/Postconditions:
// Non-Functional Requirement: Use the Merge Sort Algorithm.

```c
strncpy( order.
    of the linked list of the original C-strings sorted into
    post: The return value is the addr. of the first node
    non-empty) linked list of C-strings.
    a
    pre: Instance==the addr. of the first node of a
    node * MergeSort( node * Instance );
```
What is a tree?

A rooted tree is a structure of nodes and arcs (pairs of nodes) that has:

(1) One root node (and)
(2) Zero or more rooted trees, with no nodes or arcs in common with each other
(3) One arc from this tree's root to the root of each of the trees specified under

(c) or the root. (and)

(q)
(q), are subexpressions of the expression.
Any operator and operands under
expression overlap (and) or more expressions
or has an operator, it has one
sively (and) or has a top level operator, exclu-
Either is an identifer or constant,
An expression

under (p).
the root of each of the trees specified
(c) One arc from this tree's root to
other or the root (and)
other no nodes or arcs in common with each
trees, with
(q)

A tree has:

(c) One root node (and)
XXX employee.

Be sure to use parentheses to keep track of your calculations. Remember that multiplication and division have higher precedence than addition and subtraction. However, parentheses can be used to override the default precedence. For example:

\[ (3 + 4) \times 5 = 35 \]

\[ 3 + (5 \times 4) = 23 \]

In general, it's a good idea to use parentheses to make your code easier to read and understand.

2 Practice Rules:

1. If it's doubtfully or subtlety USE PARENTHESES to make it

\[ 4 + 5 \times 4 = 20 \]

\[ (4 + 5) \times 4 = 36 \]

3. Here's a mnemonic: Multiply, Divide, Add, Subtract. For example:

\[ a = b + c \times d - e + f + g \]

4. Parentheses are not fully parenthesized:

\[ a = b + c \times (d - e) + f + g \]

5. Expressions

To make learning these ideas easier, we will start with fully parenthesized expressions.
The top level operation "assign" a value to a variable. Why MUST it be done last? It uses the results of the all previous operations.

1. Assign it also to a.
2. Assign the last sum to b.
3. Subtract that from C, remember result.
4. Multiply f and c.
5. Add subtraction result to this last product.
6. Assign the last sum to b.
7. Multiply old value of e by d.

Increment e first.

Means:

\(((((f \cdot e) + ((d \cdot (e++) + c)) - ((B = C) \cdot (D - C))) \cdot (E++) + f) \cdot c)\)

Fully parenthesized:

\(\text{A=B=C} \cdot (E++) \cdot (f \cdot c)\)
Example of an expression and its Parse Tree

\[
(A = (B = ((C - (D * (E++))) + (F * G))))
\]
(F*G)  

The top level operator is multiplication (\star)
\[
(C - (D * (E++)))
\]

Top level operator is subtraction \((-\)\).

\[
(D * (E++))
\]

Top level operator is multiplication \((\ast\)\).

\[
(E++)
\]

Top level operator is increment \(++)\.)
An expression is a top level operator, either an identifier or constant, or more expressions as operands (and) overlaps.

(3) Any operator and operands under the root of each of the trees specified.

(4) One arc from this tree's root to other or the root. (and) no nodes or arcs in common with each other or more rooted trees, with

(2) One root node. (and)

A tree has expression(s) which the expression of the

definitions: FIT these expressions. FIT these expressions. FIT these expressions. FIT these expressions. FIT these expressions.

Your Job: Check that these examples, purporting to be "trees and expressions," are expressions.
The root node:

\[(C - (D*(E++)))\]

other trees, nothing in common.

Tree def. clause (p) is OK!

Clause (c) is OK too!

identifier

identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

Identifier

I
Clause (c) is OK.

The operands are substituting:

Expression det. Clause (b) is OK?

Not overlapping.
Expressions as operands.
Is 1 or more which

identifier

(top level operator is increment (++)

(++)

(top level operator is multiplication (*

(*

(D* (E++)

(identifier

(top level operator is subtraction (-

-)

(C-(D*(E++)))

Has an operator (a)

Clause (a)
and return its result. 

3) Combine the results from (2) using the meaning of the operator to compute of the operands. (Only one call for a unary operator.)

2) RECURSIVEELY call Evaluate(L), call Evaluate(L2) for each of the trees

identifier. So, return it or its value.

1) If L is just one node only, then the expression must be a constant or Evaluate(ParseTree L)

tree:

The following recursive algorithm evaluates an expression when given its parse

tree. The "tree of an expression" is called the expression's Parse Tree. Parsing expression is called the expression's Parse Tree. The (rather difficult and non-trivial) job of determining out the tree from a given absolute clarity. The tree of an expression represents the expression's structure with
Recursive function.

Label is to OBSERVE the stack of ACTIVATION RECORDS during the run of a (insert) into a stack is called push; delete from a stack is called pop.

and non-recursive.

other data relevant to all C/C++ function calls and returns, both recursive.

runs C/C++ programs. Implementing and organizing local variables and

3. (The "run-time stack" of activation records, internal to the system, when it

2. Storing and organizing intermediate results when evaluating expressions.

1. Figuring which pairs of parentheses MATCH in a correctly nested

3 uses for stacks:

ONLY ONE END (called the top).

that access, insertion and deletion are permitted at

What is a stack? A stack is a sequence that is restricted so

University at Albany Computer Science Dept.
5. Analyze/evaluate fully parenthesized expression 

4. Testing (simpler) string for nesting recursively.

3. Testing string using a recursive procedure, visualizing the stack of activation records. Each activation record stores one character.

2. Reversing string \texttt{BATMAN} using a stack.

1. Reversing string \texttt{BATMAN} using a stack.

At this point, we did low-tech dramatizations: recursively.

\((4\rightarrow 6-4) \times \left(3/6+9\right)\)