difference in the tree structure.

(„ordered“ means the order of the subtrees connected to the root node makes a

geraph with no circuits.

(a non-rooted) tree is a minimally connected graph, equivalently, a connected

trees. In graph theory,

in the course, we cover only finite, rooted and ordered trees.

Trees

Stacks

Expressions-Problems and Issues

CSI 310: Lecture 17
Lab 6 will be to do calculations in the stack-based postfix syntax language stacks and recursion.

Lab 5 is to introduce sequence reversal and parentheses balancing using both of a recursive function.

Lab 4 was to observe the stack of activation records during the run.

C/C++ function calls and returns, both recursive and non-recursive.

Implementation and organizing local variables and other data relevant to all runs C/C++ programs. References and pointers provide access into non-local frames.

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

expressions, such as in the "two-stacks algorithm" of Project 4.

2. Storing and organizing intermediate results when evaluating or parsing

parenthesized expression.

1. P对其进行匹配

3 uses for stacks:

- ONLY ONE END (called the top).
- (()dod) are permitted at
- ()push () and deletion

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

University at Albany Computer Science Dept.
Postscript.
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 used?

In Java, class files...

Examples of (sequences of) Expressions: Calculator input; C++/Java source

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

Problems with Expressions:

\[(\overline{z(1 - \overline{z}^2)} + \overline{z(1^{\overline{y} - \overline{z}\overline{y})} + \overline{z(1^{x - \overline{z}x})})}\]
The particular expressions input to the computer for an expression evaluator program to evaluate are NOT KNOWN AHEAD OF TIME.
which are strings, not trees.

They don’t solve the problems (1) and (2) because the user’s input expressions

But what about expression trees?

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.

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

Facts about infix expressions:

University at Albany Computer Science Dept.
4. Demonstrate tree printing and evaluation on that tree.

3. Repeat it so it builds the expression tree.

\[
\begin{align*}
\text{(3)} & : 5 \\
\text{(2)} & = 15/3 \\
\text{(1)} & = [6+(9/3)]
\end{align*}
\]

operation and (3) the value: 5

Print (1) each such subexpression before evaluating its operator, (2) the last

Locating the SUBEXPRESSON equaling that number.

2. Enhance it so each intermediate „number” also carries information

1. Implement the two-stack algorithm to evaluate infix expressions.

Project 4:
Let's begin with stacks...
recursive function.

Lab 4 is to O\textsc{b}s\textsc{e}r\textsc{v}e the stack of A\textsc{c}t\textsc{i}v\textsc{a}t\textsc{i}on \textsc{R}e\textsc{c}o\textsc{r}d\textsc{s}

\begin{itemize}
\item Insert into a stack is called \textsc{p}u\textsc{sh}, delete from a stack is called \textsc{p}o\textsc{p}.
\end{itemize}

and non-recursive.

other data relevant to all C/C++ functions calls and returns, both recursive
runs C/C++ programs. Implementing and organizing local variables and

3. (The \textsc{r}un-\textsc{t}ime stack\textsc{c} of activation records, internal to the system when it

2. Storing and organizing intermediate results when evaluating expressions.

parenthesized expression.

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

3 uses for stacks:

\begin{itemize}
\item (called the top)
\item \textsc{o}n\textsc{l}y \textsc{o}n\textsc{e} \textsc{e}n\textsc{d}
\item \textsc{a} stack \textsc{i}s \textsc{a} sequence \textsc{a}t \textsc{h}at is restricted so
\item that access, insertion and deletion are permitted at
\end{itemize}

What is a stack?
3. Testing string ({} {}) for properly nested parenthesisization using a stack (SEE DSO sec 7.1)

2. Reversing string BATMAN using a recursive procedure, visualizing the stack of activation records. Each activation record stores one character. (SEE B)
{ //
    print c; //
    print ST; //
    // Setting ST significance the rest of S,
    set first char of S in local variable c; //
} else //
    if (S is "") do nothing;
    pseudocode://
    post: the reverse of S has been printed
    pre: pcn points to a c-string S
    void printf(char *c); // Recursive reverse string printf function

University at Albany Computer Science Dept.
string user_input;
}

int main()
{
    // Given expression are balanced. Otherwise, the return value is false
    // Postcondition: A true return value indicates that the parentheses in
    // Boolean is-balanced(const string expression);
    // PROTOTYPE for a function used by this demonstration program

    using namespace std;
    string provides stack // #include <string>
    provides stack // #include <stack>
    std::cout, cout // #include <iostream>
    provides EXIT_SUCCESS // #include <cstdlib>

    // A small demonstration program for a stack.
    FILE* File=

    // GET -- no-recursive parenthesis tester. Get it with
    // University of Albany Computer Science Dept.
}
const char RIGHT_PARENTHESIS = ')
const char LEFT_PARENTHESIS = '('

// Meaningful names for constants

Library facilities used: stack, string

bool is_balanced(const string& expression) {
    return EXIT_SUCCESS;
    cout << "That ends this balancing act."
    cout << "Those parentheses are not balanced."
    else
    cout << "Those parentheses are balanced."
    if (is_balanced(user_input))
        getline(cin, user_input);
    cout << "Type a string with some parentheses:"
}
{
  return store.empty();
}

{
  failed = true;

  else if (next == RIGHT_PARANTHESES && store.empty())
    store.pop(); // Pops the corresponding left parenthesis.
  else if (next == LEFT_PARANTHESES && !store.empty())
    store.push(next);

  if (next == LEFT_PARANTHESES)
    expression[t] = i;

  for (i = 0; i < expression.length(); ++i);
}

// becomes true if a needed parenthesis is no
// boolean failed = false; // The next character from the string
// char next; // An index into the string
// string::size_type i; // Stack to store the left parentheses as the

University at Albany Computer Science Dept.
else return pch+1; //
if (pch == NULL || NotMatch(pch[0], saved) return NULL;
    pch = Establish(pch);
}
while (pch == NULL and pch[0] is a LEFT P)
    pch = pch + 1;
LOCAL Saved = pch[0]; pch = pch + 1;
if (pch[0] is not an LEFT PARENTHESES) return NULL;
assert (pch); assert (pch[0] == \\
 "\0", 0);

: pseudocode:
ELSE the return value == NULL.
BALANCED prefix.
char after the SHORTEST NON-EMPTY
THEN the return value points to the first
post: If S has a non-empty BALANCED prefix,
pre: pch points to a non-empty C-string.
char * Establish (char * pch) {

Recursive nested parentheses tester.