Trees, Recursion, Expressions, Stacks

Recursion

CSI 310: Lecture 12 (Spr 05)
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

\[
\{ \text{if}(n==0) \text{return } 1; \} \text{ else return fact(n-1)*n;}
\]

int fact(int n)

Example: A function is recursive means the body of the function sometimes,

\[
\text{Re-implementatio n it's worth it.}
\]

(3) Elegant way to write programs, performance can be improved with routine

(2) Powerful problem solving technique.

(1) Understanding, not just programming, data structures and algorithms.

Recursive, Recursion, Expressions, Stacks are closely related.
SAME TIME while the recursive function runs.

So... there will be allocated MANY DIFFERENT variables named in AT THE
activation of the function.

IMPORTANT: There is a separate local extent variable for each and every
which means something else.
Most people say "local" for short, but they get confused about "local scope"

SYNONYM for "extent".

Automataic variables are often called local extent variables. "Lifetime" is a

\[\text{in within fact}(\text{int } n) \begin{array}{l}
\{ \text{else return fact}(n-1) \} n; \\
\end{array}\]

\[\text{if}(n==0) \text{return } 1; \]

\[\text{int fact}(\text{int } n) \]
```c
{ int CAT6(int n) { steep(10000000); return -1; }

{ int CAT5(int n) { if(n==1) return 1 else return CAT5(n-1)*n; }

{ int CAT4(int n) { if(n==1) return 1 else return CAT4(n-1)*n; }

{ int CAT3(int n) { if(n==1) return 1 else return CAT3(n-1)*n; }

{ int CAT2(int n) { if(n==1) return 1 else return CAT2(n-1)*n; }

main() { int ret; ret = HATCAT(4); count >> ret >> endl; }

But let's begin without any recursion at all. With apologies to Dr. Suss,
Only true „logically”...this data is actually stored in the called activation.

**ACTIVATION is destroyed, and its Record gets recycled.**

Real & **When an ACTIVATION executes the return; operation, that**

CALLED a function

spot within the function’s body if this activation created this one. (3) The return

**DIFFERENT ONE** whose CALL operation created this one. (2) A pointer to the activation

activation’s local (i.e., automatic) variables. (1) An

**Definition of Activation Record:** The data structure that holds (1) An

will control what THAT activation does.

Wrong: Control „jumps” or „goes to“ the function’s body.

Wrong: Control „jumps” or „goes to“ the function’s body.

What HAPPENS when the computer executes a FUNCTION CALL?
3. A stack is a sequence for which insertion and deletion are only done at one end (called the top).

2. Activation records are stored in a stack (like a pile of bills).

1. Current Lab (4): Observe the activation records for runs of the recursive function write-vertex() from DS0 chapter 9.
6. Write interfaces and pre/post conditions for a split function, _merge sort().

5. Follow instructions to implement sort algorithm timing.

4. Code _select sort() in with pre and post conditions. Implement unconditional call to _print for ward() with a commanded call.

3. Command phase user interface/command input and dispatch. Replace _print for ward() in _test.h.

2. Input phase linked list builder/_print for ward() for testing/primitive.

1. Input phase link reader and _select for ward() for primitive.

Project 3 Management

University at Albany, Computer Science Dept.
7. Begin mergeSort() body. COPY OR rewrite your split code to put in
mergeSort().

8. Design, code, test and debug the function

9. Finish mergeSort() to use split and merge; test and debug.

node * merge(node * pt, node * p2){

10. Implementation mergeSort timing, perform final review of project specifications,

test, debug, final testing, project completion/turnin.
char buffer[BUFFSIZE];

Data Structure Diagram

State Diagram for sorter

Some oct4ext\0
Here are sm\0
State 12\0
str 12\0
strtxtxyzp\0

Exit

Start

Read and Process Text

Input Phase

Read and Dispatch Commands

merge\0
split
print splitting
print forward

Selection sort

print forward
return 0;
} // end

A[0..nch-1] is SORTED

NOW, A[t] has the smallest char from A[1..nch-1]

{ /* code to "swap" A[t], A[t] */

if (A[t] < A[t+1]) { nch++; t+1; } }
}

for (int t=0; t < nch-1; t++)

if (cin >> A[t] && return 1; int nch = strlen(A);

const int ASIZE = 100; char A[ASIZE];

main()

using namespace std;

#include <fstream>
#include <iostream>

Selection sort demo: Processes chars within the array A.
We will then illustrate the (recursive) MergeSort hall.

We now illustrate what Hall's Project 3 work must do.

It uses the Selection Sort algorithm for sorting.

This program manipulates characters as if they were numbers.
Cat

Aardvark

Bat

Ape

Caterpillar

Dog

Ant

Zebra

Sample list of items to sort:
How can we very efficiently swap the strings in the nodes pointed to by $I$ and $J$?
Computer DOES NOT copy chars nor node pointers!

Swap the values in the 2 data fields of the nodes pointed to by I and J.
is called "Merging"  

3. Combine the two sorted groups into one large sorted list. This combining

INDEPENDENTLY:

2. Sort each of the smaller groups (by recursive calls). That means


1. Divide the elements to be sorted into two groups of equal (or almost equal

arrays, which is easier)

becoming of DOS 13.2 BUT you will implement merge-sort on linked lists, not

paradigm or pattern is applied to the problem of sorting a sequence: (read the

Mergesort is the algorithm that is invented when the divide-and-conquer
(or p==NULL, return the sorted list holding the strings from L1 and L2 is formed).

// Post: The return value points to a linked list containing the original C-strings
// arranged in non-decr. lexicographic order.

node * MERGE (node * p1, node * p2) {
  struct node * node; char * data;
}

Please declare and code these:
Slides on trees/recursion/expressions/stacks.
(q).
(c) One arc from this tree's root to the root of each of the trees specified under
or the root, (and)

(p) Zero or more rooted trees, with no nodes or arcs in common with each other

(a) One root node, (and)

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

What is a tree?
An expression

(a) are subexpressions of the expression
(b) Any operator and operands under
(c) a level operator has one
(d) an operand, or

An expression

united by

(a) zero or more operators
(b) root node
(c) one arc from this tree's root to

other or the root

no nodes or arcs in common with each

under (p)
XXX employee.

smartness with the complex C++ precedence/associativity rules, FIRE that
obvious. (2) If a programmer you are supervising tries to show off his/her
2 Practical Rules: (1) If it's doubtful or subtle, USE PARENTHESES to make it

\[
\begin{align*}
35 & = 5 \times 7 = (5 + 4) \\
20 & = 3 + 2 \times 3
\end{align*}
\]

precedence than addition, from elementary school:

Rather/completely operator precedence and associativity rules people

How do you know which operator is evaluated first?

Not fully parenthesized:

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

fully parenthesized:

To make learning these ideas easier, we will start with fully parenthesized

University at Albany Computer Science Dept.
The top level operation, "Assign to A," is executed LAST! (Why MUST it be done last? It uses the results of the all previous operations.)

7. Assign it also to A.
6. Assign the last sum to B.
5. Add subtraction’s result to this last product.
4. Multiply F and G.
3. Subtract that from C, remember result.
2. Multiply old value of E by D.
1. Increment E first.

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

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

fully parenthesized:

means:
Example of an expression and its Parse Tree

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

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

- \(A\): top level operator is assignment (=)
- \(B\): top level operator is assignment (=)
- Identifier B
- Top level operator is addition (+)
- \((C - (D*(E++)) + (F*G))\)
- \((F*G)\)
- \((C - (D*(E++))\)
- Top level operator is subtraction (-)
- \((D*(E++))\)
- Top level operator is mult. (*)
- \(D\)
- \(E\)
- Identifier E
- (details cont’d on next 2 frames)

(cont’d on next 2 frames)
top level operator is multiplication (F*G)
The top level operator is subtraction (C - (D* (E++)))

The top level operator is multiplication ((D* (E++))

The top level operator is increment (E++)

Identifier: D

Identifier: E

Identifier: C

Identifier: H
(a) Either is an identifier or constant;
(b) If it has an operator, it has one
(c) Any operator and operands under
    or more expressions as operands (no
    overlaps!)

An expression

Your job: Check that these examples, purporting

definitions:

under (q).

the root of each of the trees specified

(c) One arc from this tree's root to
    other or the root. (and)
    no nodes or arcs in common with each

(b) Zero or more rooted trees, with

(a) One root node. (and)

A tree has:
Tree det. clause (c) is OK?
other trees, nothing in common

Tree det. clause (b) is OK?

Which IS 0 or more

Identifier: C

Identifier: D

Identifier: E

Identifier: A

Identifier: B

Identifier: E

Identifier: B

Identifier: F

Identifier: G

Identifier: H

Identifier: I

top level operator is subtraction

C - (D * (E++))

(D * (E+++))

* (E++)

top level operator is increment

(E++)

( )

top level operator is multiplication

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

bottom level operator is addition

D + E

bottom level operator is subtraction

C - D

bottom level operator is multiplication

D * E

bottom level operator is increment

E++

bottom level operator is division

D / E

bottom level operator is modulus

D % E

bottom level operator is equality

D = E

bottom level operator is inequality

D != E

bottom level operator is less than

D < E

bottom level operator is less than or equal

D <= E

bottom level operator is greater than

D > E

bottom level operator is greater than or equal

D >= E

bottom level operator is assignment

D := E

other operators
Clause (c) is OK.

The operands are substrings:

Expression del. Clause (\(p\)) is OK?

Not overlapping.

Expressions as operands:

Is 1 or more which

Has an operator:

\[ (\overbrace{\text{top level operator is subtraction}}^{\text{identifier}} \ (C-\overbrace{\text{\(D\times\text{E++}\))}}^{\text{top level operator is multiplication}}\ ) \]

\[ \overbrace{\text{\(E\)}}^{\text{identifier}} \overbrace{\text{++}}^{\text{top level operator is increment}} \]

\[ (\overbrace{\text{\(D\times\text{E++}\))}}^{\text{identifier}} \]
3) Combine the results from (2) using the meaning of the operator to compute.

(2) RECURSIVELY Call Evaluate(L1) for each of the trees. Identify the node, return its value.

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

Evaluate(ParseTree L)

The following recursive algorithm evaluates an expression when given its parse tree.

Parse Tree

The “tree of an expression” is called the expression’s structure.
A recursive function.

1. The “run-time stack” of activation records, internal to the system when it
   runs C++ programs.  Implementing and organizing local variables and
   other data relevant to all C++ function calls and returns, both recursive
   and non-recursive.

2. Storing and organizing intermediate results when evaluating expressions.

3. (The “run-time stack” of activation records, internal to the system when it
   runs C++ programs.) Implementing and organizing local variables and
   other data relevant to all C++ function calls and returns, both recursive
   and non-recursive.

A stack is a sequence that is restricted so

ONLY ONE END (called the top) that access, insertion and deletion are permitted at

3 uses for stacks:

What is a stack?