Trees, Recursion, Expressions, Stacks

Merge Sort of Linked Lists

CSI 310: Lecture 14
But, running \texttt{fact}(0) calculates \texttt{0! = 0 \ is \ true} and returns \texttt{1}.

Since \texttt{720 - 1 = 6}, \texttt{fact(6)} computes \texttt{720 \times 6 = 4320} and returns it.

For example, running \texttt{fact(7)} computes \texttt{7! = 5040} and returns it.

\begin{verbatim}
if (n==0) return 1;
else return fact(n-1)*n;
\end{verbatim}

\texttt{Example:} When the function runs, calls the same function, either directly or indirectly.

\texttt{Definition:} A function is recursive if its \texttt{it\_is\_worth\_it.} recursive implementation is \texttt{it\_is\_worth\_it.}

(3) Elegance way to write programs; performance can be improved with routine re-implementations.

(2) Powerful problem solving technique.

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

\texttt{Recursion:} Trees, Recursion, Expressions. Stacks are closely related.

 university at Albany Computer Science Dept.
how you observed this to turnin-cs1310-cslab4
recursive function variables. For a checkoff, demonstrate or submit an essay on
observe that DIFFERENT activation records have DIFFERENT copies of local
Figure out, perhaps with the help of the lab instructor and/or classmates, how to
Copy, build, and run it under the debugger.
You will find an expanded version of the "vertical" program in ~acslab4/
Read section 9.1 of the text and bring your textbook to the lab.
Lab4 Exercise:
{ return Solution; }
{ Solution = Result from COMBINING CALCULATION (Ans1, Ans2, etc.); 
 answer-type Ans1 = Solution function(Problem2); etc. 
 answer-type Ans2 = Solution function(Problem1); etc. 
 } 

// RECURSE one or more times:
// BREAK up instance into Subproblem1, Subproblem2, etc.
input-type Subproblem1, Subproblem2, etc. 

} else 
{ Solution = Result from DIRECT CALCULATION (Instance) 
}

if (Instance is small enough) 
{ answer-type Solution 
} 

answer-type Solution function(Problem) 

Divide and Conquer Pattern

University at Albany Computer Science Dept.
return Solution;

Solution = MERGE (Ansl, Ansz);

if Ansz = MERGEORT (Subproblem2)

if Ansl = MERGEORT (Subproblem1)

/* RECURSE one or more times:
Subproblem2 = Instance; /* Split removed half the original list.
Subproblem1 = Split (Instance); /*

/* TRIVIAL SORT OF A 1-ELEMENT LIST */

else

Solution = Instance; /* TRIVIAL SORT */

if (Length (Instance) == 1)

Solution = MERGEORT (Solution, Instance)

Divide and conquer pattern applied to sorting

University at Albany Computer Science Dept.
(2) The return value is the addr. of the first node of the half that was removed.

The nodes removed.

(1) Instance=variable. Value, but with approx. half

Linked list of at least two C-strings.

Pre: Instance=the addr. of the first node of a

post: Instance=split (Instance * mode) from <const>

C-strings will be compared using strcmp() from <string>

Mode is a linked-list node type; each node's data is the address of a C-string.

Merge sort explained using Pre/Postconditions:
The pre-condition for MERGESORT means the input is not sorted at all.

The post-condition for MERGESORT means most sorting work was already finished.

The post-conditions are identical. But, the pre-condition

```c
strcmp( ) order.

post: The return value is the addr. of the first node of the linked list of the original C-strings sorted into.

// Non-functional Requirement: Use the Merge Sort Algorithm.

post: The return value is the addr. of the first node of a non-empty (non-null) linked list of C-strings.

printf( "Instance=\"the addr. of the first node of a"");
```
{ return Solution;
{
Solution = MERGE (Anst, Anst);  
} 
dnode * Anst = MERGE (Subproblem2);
} 
dnode * Anst = MERGE (Subproblem1);
//RECURSE one or more times:
Subproblem2 = INSTANCE; //SPLIT removed half the original list;
Subproblem1 = SPLIT (INSTANCE);
} 
dnode * Subproblem1, Subproblem2;
} 
else
{
Solution = INSTANCE; //TRIVIAL SORT OF a 1-ELEMENT LIST!!
}
if (length(INSTANCE) == 1)
{ 
dnode * Solution;
} 
dnode * MERGESORT (dnode * INSTANCE)
Subpr1=SPLIT(Instance); Subpr2=Instance;
What is the result of executing this line of code?

```c
node * Ans1 = MergeSort( Subpr1 );
```

I don't care about ANYTHING ELSE!!!!!
computer!!! Just trust the post conditions
about HOW recursive call results were
How to think recursively: DON'T WORRY
I Don't Care about ANYTHING ELSE!!!

What is the result of executing this line of code??
dnode * Ans1 = MergeSort( Subpr1 );

Ape

Bat

Aardvark

Cat
MergerSort(Subpr2)

Ans2 =

RESULT

what is the

Subpr2
The result of the recursive call is to make the list sorted properly!

\[ \text{Ans2} = \text{MergeSort}(\text{Subpr2}) \]

Diagram: [Diagram of a sorted list with nodes labeled "Caterpillar", "Dog", "Ant", and "Zebra".]
Let’s complete the job: First, redraw the diagram…

```c
MergeSort(Subpr2);
Ans2 = dnode * Ans1 = MergeSort(Subpr1);
```

```
Cat
NULL
Aardvark
NULL
Bat
NULL
Ape
NULL
```

```
Zebra
NULL
Ant
NULL
Dog
NULL
```

```
Caterpillar
NULL
```

```
MergeSort (Subpr2)!
Ans2 = dnode * Ans1 = MergeSort (Subpr1) !
```
these two sorted lists.

Merge

Combination operation:

MergeSort(Subpr2);

Ans2 =

dnode * Ans1 = MergeSort(Subpr1);

Bat

Ape

Aardvark

Cat

NULL

Dog

Caterpillar

Ant

NULL

node * Ans1 = MergeSort(Subpr1)!
move all remaining list1 or list2 entries to end of output list;

} // end of while loop

{
move the first entry from list2 to end of output list;
}

else
{
move the first entry from list1 to end of output list;

} // (the first entry in list1 is before or equal to the first entry in list2)

if there is a first entry in list2

else (there is a first entry in list1)

while list1 != empty list

MERGE (sorted list1, sorted list2) algorithm:

University at Albany Computer Science Dept.
A rooted tree is a structure of nodes and arcs (pairs of nodes) that has:

What is a tree?

(c) One arc from this tree's root to the root of each of the trees specified under

or the root.

and

Zero or more rooted trees, with no nodes or arcs in common with each other.

(a) One root node.

and

(b) One or more rooted trees.
An expression:

(a) Either is an identifier or constant;

(b) An operator and operands under

(c) Any operator and operands do not

overlap.

(d) At most zero or more expressions

(e) One arc from this tree’s root to

another or the root.

(f) No nodes or arcs in common with each

(g) One or more rooted trees, with

(h) A tree has

under (q).
XXX employees.

smartness with the complex C/C++ precedence/associativity rules. FIRE that
obvious.

2 Practical Rules: (1) If it's doubtful or subtle, USE PARENTHESES to make it

\[
\begin{align*}
\text{not } (3 \times 5) 
\end{align*}
\]

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

precedence than addition. From elementary school, multiplication has higher
must memorize or look up. They begin with "multiplication has higher
Rather complicated operator precedence and associativity rules people

How do you know which operator is evaluated first?

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

Not fully parenthesized:

expressions.

To make learning these ideas easier, we will start with fully parenthesized
The top level operation assigns to A is executed last. Why must it be assigned to A, also to B.

5. Add subtraction 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.

means:

\(( ((C * F) + (E++ * D) - C)) = (A)\)

Fully parenthesized:

\(A = B = C - D * E++ + F * G\)
Example of an expression and its Parse Tree:

\[ A \equiv (B \equiv (C - (D \times (E + F))) + (F \times G)) \]

Details cont'd on next 2 frames.
The top level operator is multiplication (\( \star \)).

```plaintext
(F*G)
```

identifier

identifier

\[ G \]

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

- Top level operator is subtraction (-)
- Top level operator is multiplication (*)
- Top level operator is increment (++)

Identifier: D
Identifier: E
Identifier: C
Definitions: “Trees and expressions” are these expressions:

Your job: Check that these examples purporting to be “trees and expressions” are expressions. Each of these expressions has:

1. A root node.
2. Zero or more rooted trees, with
   (a) One root node. (and)
   (b) Zero or more rooted trees, with
   (c) One arc from this tree’s root to
      (d) Other or the root. (and)
   (e) One arc from this tree’s root to
      (f) Other or the root. (and)
3. No nodes or arcs in common with each
4. Either is an identifier or constant;
5. Either is an identifier or constant;
6. Any operator and operands under
   (and) overlaps!
7. No more expressions as operands
8. Any operator and operands under
   (and) overlaps!

An expression:

This are subtrees of the expression.
(C-(D*(E++)))

top level operator is subtraction

(D*(E++))

top level operator is multiplication

(E++)

identifier
++

identifier
*

identifier

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

The root node:

IS 0 or more

which

other trees, nothing in common.

is OK?
clause (a)

Tree der.

is OK?
clause (b) is OK?

identifier

identifier

identifier

identifier

identifier

identifier

identifier

identifier

identifier

identifier

identifier
The operands are substituting: (C-(D*(E++)))

Expression det. Clause (b) is OK!

Has an operator: (C-D*(E++))

Expression det. (a) is OK!

Is it or more which expressions as operands,

Not overlapping.

Clause (c) is OK.
and return its result.

(3) Combine the results from (2) using the meaning of the operator to compute

(2) RECURSIVEELY: Call Evaluate(L1) 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

Evalue(L) result.

The following recursive algorithm evaluates an expression when given its parse

tree: P

The tree of an expression is called the expression's parse tree. P

Expression is called Parsing. P

The (rather different and non-trivial) job of parsing out the tree from a given

expression with absolute clarity.

The tree of an expression represents the expression's structure.
recursive function.

Lab 4 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

32

University at Albany Computer Science Dept.
or composed of its individual variables, taken together.

It is useful to consider the WHOLE ARRAY as ONE VARIABLE that is formed.

\[ C/C++ \text{ the indices range from 0 to length-1.} \]

- Each element is selected for access using an integer, called an index. In
- Each address, like a row of houses on one city block.
- The elements are located contiguously in memory, at adjacent
- The number of elements (length of the array) is fixed.
- `char`, `any other type`…
- Each individual variable, called an element, has the same type (int, float, `plural` (plural) variables)

Arrays, again.
to add vector A and W:

Mathematical vectors, e.g., coordinates of points in 3 dimensions. C++ code

Depending on how you tell WHERE THE END IS.

... holds strings up to 99 or 100 chars long.

char MSTRING[100];
declared:

One way to store/process string data is to use a C++ array of char.

This is called string data.

See "RIGHT HERE": R, I, C, H, T, etc.

Contents of a word processed term paper. The text on a Web page, what you

What can you use arrays for?

University at Albany Computer Science Dept.
This prints each number and price, separated by 6 spaces:

```c
{ count >> Practice[i] >> Practice[i] >> Practice[i] >> Practice[i] >> Practice[i] >> Practice[i];
    count >> } //for i=0;i<100;i++;
}
```

```
for( int i=0; i<100; i++ );
```

Stock number `Practice` and store them in `Practice[]`. //from the Internet and store them in `Practice[]`.

Get prices of stocks numbered 0 to 99

```
float Practice[100];
```

Statistics such as prices of 100 different stocks:

```
{ for( int i=0; i<100; i++ ){ +13; } for( int i=0; i<100; i++ ){ +13; } for( int i=0; i<100; i++ ){ +13; } for( int i=0; i<100; i++ ){ +13; }
```
C-strings are different from C++ strings you get from #include <string>


with:

The C-string "ABCD" (4 letters) is stored in a `LENCHT 5` (five, not 4) character.

Strings in `char` array terminated with `\0` are called C-strings.

The null `char` is coded `\0`.

In C/C++, the `char` `\0` is coded `A`.

Called the "null character".

One way to tell where the end of a string is: just after the last element used for
cout << MYCHARRAY << endl;

printing what you typed:

cin.getline(MYCHARRAY, 12);

Reading up to 12 characters you type on one input line:

holds a c-string with length up to 11

char MYCHARRAY[12] = "";

REQUIRED in CS1310: Declaring a variable that can hold a C-string:

cout << "Hello World" << endl;

c-strings are very easy to use. You have used them in CS1201 code like:

using namespace std;

#include<iostream>
{ /*
 while (!finished) { getLine: getLine failed or empty string was read.
   count ++; } /* end */ a sorted string was printed.
   now, A[0].nch-1 is sorted

   now, A[i] has the smallest char from A[i].nch-1

   /* code to "swap" A[i],A[i] */ }
   (i[0] < [0]i[1])
   for(j = i+1; j < nch; j++)
   for(i=0; i < nch-1; i++)
   ( j = nch = string A
    while(cin >> A[0].size) { /* A has size */
      count int A.size = 100; int nch, i; char A[A.size];

      using namespace std;
      main()
        #include <cstring>
        #include <iostream>
        #include <sstream> // Selection sort demo: processes char within the array A.
For next week, after the midterm, The other half is to implement the (recursively) MergeSort algorithm, a topic in computer science. We will illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.

We now illustrate what half your Project 2 work must do.
Sample list of items to sort:

- Cat
- Hardeark
- Bat
- Ape
- Caterpillar
- Dog
- Ant
- Zebra
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.

pT = I->data; I->data = J->data; J->data = pT;
and finally d.

printf the characters in order, H, then e, two I's, an o, a space, then w, o, r, t, c.

cout >> "Hello World"

is easier to think about than

printf the string Hello World

cout >> "Hello World"

C-string:

array of char (sequence of char variables) is a single variable that holds ONE

The examples of simple C-string use illustrate the usefulness of thinking that an
return 0;
}  // end of main()
function

cout << "Input from cin failed. exiting" << endl;

{ ... 

   // process the input somehow...

   (DEBUG) cout >> input >> endl;

   if (DEBUG) cout << input << endl;

   while (cin.getline(input, INBUF_SIZE))
   {
     char input[INBUF_SIZE] = 0;
     const int INBUF_SIZE = 12;
   }

   main

   #define DEBUG 1
}

using namespace std;

#include <string>
#include <iostream>
#include <iostream>

Skeleton main function for some CS130 projects:
count >> "Ent. Unknown command, try agen."
        } } else if (strcmp(input, "quit") == 0) { count >> "Thx is a won der ful pr o gram."
        } else if (strcmp(input, "about") == 0) { count >> "Ent. H e l p" { (DEBUG) count >> input } >> (cin >> input >> INBUFSIZE) >> [char input[INBUFSIZE]] >> int INBUFSIZE = 12; } main ()

Recognizing one-line commands:

University at Albany Computer Science Dept.
// end of main() function.
return 0;

cout << "Input from cin failed. Exiting" << endl;
}