Arrays, Sorting, New Project
Trees, Recursion, Expressions, Stacks
Deep versus shallow copy.

CSI 310: Lecture 12
```cpp
// printed Sap{y Happy

count >> arr{ary{c{h} = 'R'; // Modified the COPY, not the ORIGINAL.

strcpy(*pointer, arr[0]); // We made a DEEP COPY

pointer = new char[strlen(arr)];

}]

count >> arr{ary{c{h} = 'S'; // We made a SHALLOW COPY and MODIFIED it through a pointer.

pointer = arr; // printed Happy

char arr[0] = "Happy"

}()

#include <string>
#include <iostream>
```
somebody codes assignments and call-by-value constructor so YOU programmers can program deep copying to happen when
Main and Switch demonstrate overloaded operator() and the copy

Disadvantages of deep copies: The copy uses memory additional space.

Pitfall of shallow copies: Modifying the "copy" changes the original.

or fields.

A deep copy is made by allocating a new object for EACH object pointed to
pointer variables point to the SAME OBJECT.

A shallow copy is made by just copying the pointer values, so for each, TWO
pointer fields.

Suppose you have a pointer variable or a structure containing one or more
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 = 6, calls fact(6), which returns 720.

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

Definition: A function is recursive means the body of the function sometimes

Example: When the function runs, calls the same function, either directly or indirectly.

Re-implementation it is worth it.

(3) Efficient way to write programs; performance can be improved with routine

(2) Powerful problem solving technique.

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

Recursion:

Trees, Recursion, Expressions, Stacks are closely related.
A rooted tree is a structure of nodes and arcs (pairs of nodes) that has:

- Zero or more rooted trees, with no nodes or arcs in common with each other.
- One root node (and)
- One arc from this tree's root to the root of each of the trees specified under
An expression: (q). (p) are subexpressions of the expression.

Any operator and operands under (c) overlaps (and) no more expressions as operands (no
(or has an operator, it has one
(or has a top level operator, except
Either is an identifier or constant, (a) either
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 nodes or arcs in common with each
(q) zero or more rooted trees, with
(a) One root node. (and)

A tree has:
XXX employees.

smartness with the complex C/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

\[ 35 = 5 \times \frac{7}{3} \]

not \( (3 + \frac{4}{3} \times \frac{3}{5} = 20 ) \]

\[ 33 = 5 + 3 \times 4 \]

precedence than addition. From elementary school:

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?

\[ 4 = b = c - d \times e + f \times g \]

Not fully parenthesized:

expressions.

To make learning these ideas easier, we will start with fully parenthesized
The top level operation "assign to A" is executed LAST! Why MUST it be done last? It uses the results of all previous operations!

7. Assign it also to A.
6. Assign the last sum 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:

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

fully parenthesized:

\[ A = B = C - D * E + F * G \]
Example of an expression and its Parse Tree
The top level operator is multiplication ($\ast$)
The expression $(C-(D*(E++)))$ has the following parse tree:

- Top level operator is subtraction ($-$)
- $C$ is an identifier
- $+$ is an increment operator
- $D$ is an identifier
- $*$ is a multiplication operator
- $E$ is an identifier
- $+$ is an increment operator

The tree structure shows the order of operations as follows:
1. Increment $E$ to $(E++)$
2. Multiply $(D*(E++))$
3. Subtract $(C-(D*(E++)))$
An expression (q). are substrings of the expression (q). Any operator and operands under (c) and (and) overlap or more expressions as operands (no overlap). If it has an operator, it has one expression of (q). If it has an operator, it has one expression of (and) (and) -ively. (and) -ively. (and) -ively. (and) -ively. (and) -ively. (and)

Definitions: ET THESE {trees and expressions} FILL THESE

Your job: Check that these examples, purporting

University at Albany Computer Science Dept.
The root node: 

\((C-\text{(}\text{D} \times \text{E}++)\text{)})\)

Tree der. Clause (c) is OK!

Clause (c) is OK?

Tree der. Clause (b) is OK!

other trees, nothing in common.

IS 0 or more
d

identifi

E

top level operator is increment (++)

C

top level operator is subtraction (-)

\((D \times (E++)())\)

\((E++)\)

top level operator is multiplication (*)

\((++)\)

identifi
The operands are substituting:

Expression det. clause (p) is OK!
Expression det. clause (c) is OK.

Has an operator:

Is 1 or more expressions as operands.

Which top level operator is increment (+ +)

(top level operator is multiplication (*)

Is OK?

Clause (a)

Expression det.

Has an operator:

Expression det.

top level operator is subtraction (-)

( ( (E++) )

identifier

D

( (E++) )

identifier

E

(d (E++))

identifier

C

top level operator is multiplication (*)

( ++

E )

identifier
and return its result.

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

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

\[ \text{Evaluate}(\text{ParseTree}(L)) \]

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 is called the expression's structure with absolute clarity.

The tree of an expression represents the structure of the expression as an expression tree.
Recursive function.

Lab 4 is to OBSERVE the stack of ACTIVATION RECORDS during the run of a

procedure.

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.

parenthesized expression.

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
or composed of its individual variables, taken together.

It is useful to consider the WHOLE ARRAY as ONE

VARIABLE that is formed.

- C/C++: the indexes range from 0 to length-1.
  - Each element is selected for access using an integer, called an index. In
    addresses, like a row of houses on one city block.
  - The elements are located contiguously in memory, at adjacent
    locations.
  - 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)

An array is a sequence of variables.

Arrays' again.
What can you use arrays for?

•

depending on how you tell where the end is.

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, G, H, T, etc.

contents of a word processed term paper. The text on a web page, what you

something new re. CS1220 (Non-numerical "text" data, such as the

•

Mathematical vectors, e.g., coordinates of points in 3 dimensions. C++ code
to add vector \( \mathbf{v} \) and \( \mathbf{w} \):

\[
\text{double } \mathbf{v}[3], \mathbf{w}[3], \text{ sum}[3];
\]

count >> "Type vector \( \mathbf{v} \):";

count >> "Type vector \( \mathbf{w} \):";

count >> \( \mathbf{v} [0], \mathbf{v} [1], \mathbf{v} [2] \);

count >> \( \mathbf{w} [0], \mathbf{w} [1], \mathbf{w} [2] \);

\[
\text{for } i = 0 \text{ to } 2 \\
\text{sum}[i] = \mathbf{v}[i] + \mathbf{w}[i];
\]

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

\[
\text{char mystring[100];}
\]

declared:

array of char,
This prints each number and price, separated by 6 spaces:

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

Get prices of stocks numbered 0 to 99 from the Internet and store them in Practice[].

Statistics, such as prices of 100 different stocks:

```c
for (i=0; i<100; i++)
{
    [1]M << Practice[i];
}
```

```c
for (i=0; i<100; i++)
{
    Sum += Practice[i];
}
```

```c
for (i=0; i<100; i++)
{
    [1]M << Practice[i];
}
```

University at Albany Computer Science Dept.
C-strings are different from C++ strings you get from #include <string>


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

Strings in C language arrays terminated with \0\0 are called C-strings.

The null character is encoded \0\0 in C/C++ and the char \A\ is coded \A\ in \A\.

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

Printing what you typed:

// Reading up to 11 characters you type on one input line:
\n\n\n// Holds a C-string with length up to 11

\n\nchar MyCharacter[12];

// Required in CSI310: 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 CSI201 code like:

#include <iostream>

using namespace std;
```cpp
// Selection sort demo: processes chars within the array.

return 0;
}

while (finished: get line failed or empty string was read.
    cout >> A >> endl; // A sorted string was printed.
    NOW, A[0..nch-1] is SORTED

    // NOW, A[1] has the smallest char from A[1..nch-1]
    {
        /* [?] A[1], A[?] */
        
          for ( j = 1; j < nch; j++)
          
          for ( i = 0; i < nch-1; i++)
            ( A[nch = string( A[?])] )
          }
        }
    }

while (cin.getline( A, A.size(), '0')

    const int A.size() = 100; int nch, j; char A[A.size];
}

using namespace std;

int main()
```

```
For next week, after the midterm, the other half is to implement the (recursive) **MergeSort** algorithm, a topic we now illustrate what half your Project 3 work must do.

We sort the lines in the `main.h` file lexicographically, as C-strings. Project 3 consists of implementing two sorting algorithms and applying them to Project 3 consists of implementing two sorting algorithms and applying them to Project 3 consists of implementing two sorting algorithms and applying them to Project 3 consists of implementing two sorting algorithms and applying them to...
Sample list of lines 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.

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

prints the characters, in order, H, then e, two T's, an o, a space, then W, o, r, I,

\texttt{count} \texttt{\gg} \texttt{"Hello World"};

is easier to think about than

prints the string \texttt{Hello World};

\texttt{count} \texttt{\gg} \texttt{"Hello World"};

\texttt{C-string}.

array of \texttt{char} (sequence of \texttt{char} variables) is a single variable that holds \texttt{ONE}

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

} // Input from cin failed, exiting

{ cout >> "Input from cin failed, exiting"

... // Process the input somehow...

} // DEBUG

} // if (DEBUG)

while (cin >> input)

char input

const int INPUTSIZE = 12;

{ main()

#define DEBUG 1

using namespace std;

#include <string>

#include <iostream>

#include <iomanip>

Skeleton main function for some CSCI10 projects:

University at Albany Computer Science Dept.
cout >> "En? Unkown command, try again." >> endl;
}

else each previous strcmp() returned non-zero.
{
    return 0;
}

cout >> "Thanks, goodbye." >> endl;
{
    if (strcmp(input, "quit")
    {
        cout >> "This is a wonderful program." >> endl;
    }
    from <cstring>
    if (strcmp(input, "about")
    {
        cout >> input >> endl;} // DEBUG)
    if (cin >> gettine[quit, INBUFSIZE])
        char input[INBUFSIZE] = 12;
    const 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;
{
    
}