arrays

c++ compiler will initialize an array with zeros when values are not specified

he's talking about how to traverse arrays and how to display them, but this is very basic

void displayArray(int x[], int len)

{

for(int i = 0; i < len; ++i)

{

std::cout << x[i] << '\t';

}

}

now he's talking about pass-by-value and pass-by-reference

using <algorithm> to sort arrays

sort(arrayName, arrayName + arrayLength);

now he's talking about function overloading

function templates

- used to condense a bunch of overloaded functions with the same number of arguments and logic

\* ex: if you have 4 different displayArray functions each for different variables

whose headers look like:

void displayArray(int x[], int len)

void displayArray(float x[], int len)

void displayArray(char x[], int len)

void displayArray(string x[], int len)

- template <class T>

void display(T x[], int len)

\* data type T is an abstract variable we defined

how to trace functions

given:

void f(int& a, int b)

{

a++;

b+=2;

}

void g(int a, int& b)

{

a\*=2;

b--;

}

int main()

{

int x = 5, y = 7;

f(x, y);

g(x, y):

std::cout << x << '\t' << y;

}

functions in <cmath>

sqrt(x)

pow(x, y) x^y

fabs(x) |x|

exp(x) e^x

log(x) ln(x)

log10(x) log(x)

floor(x) round down x

ceil(x) round up x

functions of <iostream>

isspace(c)

isdigit(c)

isupper(c)

islower(c)

isalpha(c)

isalnum(c)

x = toupper(c)

x = tolower(c)

rand() returns an int between 0 and MAX\_INT

functions of <iomanip>

setw(int)

setprecision(int)

setfill(char)

left

right

given:

string name1 = "John", name2 = "Alexander";

float income1 = 25.30, income2 = 1234.56;

and we want a format that looks like:

John...........................25.30

Alexander....................1234.56

we do:

cout << setfill('.');

cout << fixed << showpoint << setprecision(2);

cout << left << setw(10) << name1 << right << setw(8) << income1 << endl;

cout << left << setw(10) << name2 << right << setw(8) << income2 << endl;

learning read/write to file using <fstream>

fstream x;

x.open(filename, ios::param) where param is in/out/app

while(!eof) or if(x.fail())

The Stack

FILO - First in Last Out - Just Like MtG

command effect

push add new item to the top of the stack

pop remove / get exisiting item from the top of the stack

class STACK

{

private:

string contents[5];

int counter; // counter should always be pointing to the

// next element, not current element

public:

void clearStack() { counter = 0; }

bool isStackEmpty()

{

// we must make this check BEFORE we POP something from

// the stack

// we don't want to pop if there's nothing in the stack

if (counter == 0) { return true; }

retun false;

}

bool isStackFull()

{

// we must make this check BEFORE we PUSH someing to the

// stack

// we don't want to push something if the stack is full

if (counter == 5) { return true; } // 5 as defined by

// our stack array

// 'contents' in

// private

return false;

}

void pushStack(string s) // string because our stack is type

// string according to array

// 'contents' in private

{

contents[counter++] = s;

}

string popStack()

{

counter--;

return contents[counter];

// this won't actually delete the top item of the stack,

// but we essentially treat it as if it is

}

}

int main()

{

string b[5] = {“MATH”, “PHYS”, “CMPS”, “BIOL”, “HIST”);

STACK s;

s.clearStack()

for(int i = 0; i < 5; ++i)

{

s.pushStack(b[i]);

}

while(!s.isStackEmpty())

{

cout << s.popStack() << “, “;

}

return 0;

}

Output: HIST, BIOL, CMPS, PHYS, MATH,

The problem with the above example is that this ONLY works for type string with a stack size 5.

we can get arround that by using templates

template<class T, int n>

class STACK

{

private:

T contents[n];

int counter;

public:

lorem ipsum

}

int main()

{  
 int a[3] = {3, 4, 5};

char b[4] = {‘l’, ‘o’, ‘v’, ‘e’};

string c[5] = {“mon”, “tue”, “wed”, “thur”, “fri”};

STACK<int, 3> intStack;

intStack.clearStack();

fillStack(intStack, a, 3); // ambiguous function

printStack(intStack, 3); // ambiguous function

STACK<char, 4> charStack;

etc etc

}

if, for example, we wanted to display all the uppercase and lowercase letters in a string “Today Is Monday” while keeping the same order and using stacks, we would use a temporary stack to re-reverse the order.

if we only had one stack, our output would be like:

upper: M I T

lower: y a d n o s y a d o

which is backwards because that’s how a stack works (FILO). but if we use a temporary stack, we can push to it to reverse the order again, thus giving us the original order:

upper: T I M

lower: o d a y s o n d a y

Stacks Cont. + Recursion

to calculate the nth fibonacci number using recursion

long int fibonacci(int n)

{

if (n==1 || n==2) return 1;

else return fibonacci(n-1) + fibonacci(n-2);

}

to calculate how long these algorithms take, we can use the system clock

<include ctime>

int main()

{  
 int start = clock();

fibonacci(50);

int stop = clock();

duration = stop – start; // in milliseconds

}

to estimate the runtime of any algorithm without implementing and running it, we find the big O notation of it.

n! == O(n^n)

Prefix, Infix, and Postfix

most compilers only read and understand things in postfix. while some do use prefix, the majority uses postfix.

writing a program to compute postfix expressions

ex: x = 5 – 3

in pseudocode . . .

x = 5 3 -;

string operation = “5 3 –“;

STACK<int, 2> s;

for char in operation:

if char is int:

s.push(char)

elif char is operand:

a = s.pop()

b = s.pop()

s.push(a <operand> b)

Linked List

(i missed a day that covered basics of this)

a linked list consists of data structures that hold a value and the location of the next value of the

list ending with a pointer to null (to know we’ve hit the end of the list)

NOTE: professor uses info instead of value but i’m using the latter for clarity

creating a stack using linked lists:

struct node

{

int value;

node\* next;

};

node\* list = null;

node p = new(node);

p->value = 3;

p->next = list;

list = p;

to make this more OOP friendly

class STACK\_LIST

{  
private:

struct node

{

int value;

node \*next;

};

node \*stack;

public:

STACK\_LIST()

{

stack = NULL;

}

void push(int x)

{

node \*p;

p = new(node);

p->value = x;

p->next = stack;

stack = p;

}

void display()

{

node\* p = stack;

while(p != NULL)

{

cout << p->value << ‘\t’;

p = p->next;

}

}

};

using generic linked lists (not the stack one), how do we insert something in the middle?

node \*prev, \*current;

while(next->value != value\_we\_want\_to\_place)

{

idk. basically we update prev and current pointers to keep up with the list so we can update the nodes when we’ve gotten to where we want to be

here’s what the prof. is typing up

class LINK\_LIST

{

private:

struct node

{

int info;

node \*next;

};

node \*list;

public:

LINK\_LIST() { list = NULL; }

void pushStack(int x)

{

node \*p = new(node);

p->info = x;

p->next = list;

list = p;

}

void displayList()

{

node \*p = list; // do this because we’ll be traversing

while (p != NULL)

{

cout << p->info << ‘\t’;

p = p->next;

}

}