====================================================

Block 2:

SUMMARY: demo binaries on Buffy

WHEN I SHOW YOU AN EXAMPLE OF THE PROGRAM BEING RUN, I USUALLY

REFER TO THE PROGRAM AS "a.out" I do this because "a.out" is

the name command-line compilers default to on big systems.

If you compile asst3b.cpp:

g++ asst3.cpp

...and there are no errors, the compiler will create a new

file:

a.out

To run the progam, you just type its name at the command (dollar

sign) prompt:

$ a.out

To run a program from CodeBlocks, press the "play tape" button,

or click debug->run.

To store your work on Buffy, read:

http://209.129.16.61/~hhaller/data/cisc192/modules/using\_psftp.exe.txt

Follow directions carefully.

READ CHAPTER 2, Storing Data in C++

Variables are names for places in the computer's memory.

The language keeps track of them in intelligent ways.

Variable types:

C is case sensitive. test and TEST are two different identifiers.

logical bool

character char

integer int

floating point float

double floating point double

valueless void

All variables must be declared prior to use. The general

form is:

type variable\_name ;

float f ;

int y ;

char ch ;

Arrays

type variable\_name[(int) how\_many] ;

int month\_totals[12] ;

int department\_hours[300] = {0} ; // set all elements to 0

Structures: collections of variables grouped under one name.

struct struct\_name

{

type element1 ;

type element2 ;

type element3 ;

} struct\_variable ;

Unions: multiple variables share the same memory:

union union\_name

{

int i ;

char c[4];

} u ;

Enumerations:

A list of values which may be assigned to the variable.

enum cities (Houston, Austin, Amarillo} ;

enum cities c ;

c = Houston ;

Unless otherwise initialized, in the above example,

Houston is a const for 0, Austin for 1, Amarillo for 2 ;

Storage classes:

extern

auto

register

const

volatile

static

Scope and lifetime of variables:

Two kinds: local and global.

globals are defined outside any function.

Within a function, you "declare" a global variable with

extern. So if it's defined in a different file, the

compiler won't flag an error while compiling this one.

The Linker handles tasks the Compiler postpones like

this one.

int g ;

void flip(void)

{

extern int g = 0 ;

}

locals are defined after the opening brace of a function.

Every program must have a function int main(), that is where

execution begins.

Other functions may have variables which only exist when the

function is running:

int f2(void)

{

int y = 0 ;

.

.

.

}

Y is created and destroyed as the function begins and returns.

BUT: static variables last as long as main() is running:

int f2(void)

{

static int y = 0 ;

y++ ;

return y ;

}

Y will only be initialized once, it will hold its value

between invocations of f2().

auto is the opposite of static, and is the default.

register asks the system to dedicate one hardware

register to the variable, making it faster to access. There

is no guarantee it will do so.

for (int outer = 0 ; outer < 40 ; ++outer)

{

for (register int inner = 0 ; inner < 500 ; ++inner)

{

.

.

That's it for the basic types, anything after arrays are

"user defined types" implimented as OBJECTS, and we'll

come to them in a while.

For now, know what to pick, and how to manipulate it.

Variables are claims the program stakes on system memory.

Once you "define" a variable, you own that place in memory

and can store or retrieve data there. If you try to

reference memory you don't own, modern Operating Systems

will abort your program. Old ones like MS-DOS will permit

it, but you may crash the system if you don't know what

you're doing.

Computers were built for handling integral types.

Whole numbers are exact, where floating point (real)

numbers are not.

Some operations on variables:

int months = 12, weeks = 52, i ; // allocate three variables and initialize two

x = y = z = 0 ; // set all three to 0

total = x + 12 ; // assign a sum

count++ ; // add 1 to count

count = count + 1 // add 1 to count

count += 17 ; // add 17 to count

count-- ; // subtract 1 from count

-------

given:

int a, x = 0 , y = 12 ;

then:

x = ++y ; // x is 13, y is 13

a = x++ ; // a is 13, x is 14

For a year where Jan 1 is a Sunday, and Sunday is 0, Mon = 1, etc.,:

int julian\_date = jdate() ;

int day\_of\_week = jdate % 7 ;

'%' is the modulus or remainder operator.

jdate may have any number 0 to 365 in it if it's a leap

year, so if we divide jdate by 7, throw away the quotient

and keep the remainder, that is the number of today's day

of the week. Think about this until you see it.

-------

See the operators table in the book: "What is that symbol?"

Braces { and } create blocks of code.

#include <file.h> // preprocessor directives include source code for compilation

string type: read carefully

Boolean variables.

cout << console output

cin >> console input

Conditional Operator

PRACTICE WITH CODEBLOCKS, LINUX, and/or VISUAL STUDIO.

UNDERSTAND HIERARCHY OF OPERATIONS IN C/C++:

(remember: "ARITHMETIC RELATIONAL ASSIGNMENT")

(The arithmetic operations like "x + 5 / 2" complete first,

then relational terms like x > y ,

then assignments "x = 89 ; ".

So:

x = (y < 37 + 1) ;

is a possible statement. Ugly, though, and unclear, and the only

possible values of x are 0 and 1. Do you see this?

True is 1, False is 0, so any expression with '<' in it

is either True or False, therefore x can only be 1 or 2.

Confirm this to yourself.

(37 gets added to 1 producing the temporary expression 38

the 38 gets compared with the value in y

the result, true (1) or false(0) gets stored in x.

Arithmetic order:

[PARENTHESES], [MULTIPLY AND DIVIDE], [ADD AND SUBTRACT]

UNDERSTAND BASICS OF IOSTREAM "<< AND >>" INPUT/OUTPUT IN C++

END SUMMARY

==========================================

A VERY important item for your notes is the Order in Which Things Happen:

C Hierarchy of Execution

() [] left to right "group indicators"

! ~ ++ -- + - \* & (type) sizeof right to left "unary operators"

\* / % left to right "multiplication"

+ - left to right "addition"

<< >> left to right "bit shifting"

< <= > >= left to right "inequality comparisons"

== != left to right "equality comparisons"

& left to right "bitwise AND"

^ left to right "bitwise XOR"

| left to right "bitwise OR"

&& left to right "logical AND"

|| left to right "logical OR"

?: right to left "conditional operator"

= += -= \*= /= %= &= ^= |= <<= >>= right to left "assignment ops"

, left to right "comma"

PRINTING FROM AND READING INTO VARIABLES:

Read: <A HREF="http://209.129.16.61/~hhaller/data/cisc192/modules/iostream\_one">iostream\_one</A>

HACKERS ONLY: Some useful information about variables on your

system can be found in the "limits.h" file.

<A HREF="http://209.129.16.61/~hhaller/data/cisc192/modules/limits.h.txt">limits.h</A>

Variables have TYPE. We're seeing integers, characters, strings, and

Boolean types in this chapter. Type specifies a) how you plan to use that data,

and b) how many bytes the data occupies.

======================EXAMPLE PROGRAMS FROM THE TEXT==========================

Memorize this program, you will use it over and over:

1.5 CHARACTER I/O (Input and Output)

//Waterpump.cpp

#include <stdio.h>

main()

{

int c ;

c = cin.get() ;

while (!cin.eof()) // line 7

{

cout.put(c) ;

c = cin.get() ;

}

}

Note: line 7, cin.eof(), if it's true, means that all

the input has been read. '!' means 'not'. cin.get()

reads a single character from the input. cout.put(x)

prints the character contained in x on the output.

1.6 ARRAYS

Arrays are ways of grouping the same kind of values

together under a single name.

int months [12] ;

int days [7] ;

The individual elements in an array are referred to with

an index:

days[0] = 1 ;

days[4] = 57 ;

Important: arrays begin at 0. If you have an array: int x[20] ;

then it has elements x[0] to x[19] NO ELEMENT x[20] !)

In programming we start counting with 0, not 1.

The way we process arrays is by using for() loops:

int i, a[10] ;

(... input or calculation)

for (i = 0 ; i < 10 ; ++i)

{

printf("a[%d] == %d\n", i, a[i]) ;

}

1.9 Character Arrays

In C++ you hear a lot about string objects. That comes later. For now,

we're dealing with the simpler Array of char. (At the heart of a C++

string Object is still an array of char, so there's really only one

way to store this kind of data on the computer.)

We'll introduce a new function: fgets()

1 /\*

2 char arrays

3 \*/

4 #include <stdio.h>

5

6 int main()

7 {

8 char buffer[81] , i ;

9

10 printf("Type your first, middle, last names: ") ;

11 fgets(buffer, sizeof(buffer)-1 , stdin) ;

12 puts("You typed: ") ;

13 puts(buffer) ;

14 }

The fgets() function will read in multiple words, even when separated

by spaces, and it needs three arguments:

what buffer (character array) it should store into

how many characters maximum it is allowed to try an store

what file it is to read the input from

The first is buffer[], and the second is the size of buffer[]

minus one. That way, the user can type in the maximum possible

size, but one space will be left for a trailing NULL. (Char

arrays must end with a NULL so other functions can process them

properly.)

The last is "stdin". Every C/C++ program begins with three i/o

streams open:

These are the C versions:

stdin

stdout

stderr

These are the C++ "stream i/o" versions: (We'll learn them later.)

cin

cout

cerr

Just remember that you must give a function what it needs to do its job.

Functions only "know" what they can "see", and if it's not in their

arguments, and not hard-coded in their bodies, they can't know it.

This is called "variable's scope"

1.10 External Variables and Scope

Scope is a simple notion: where can a variable be seen.

Variables defined inside functions can only be seen in that function.

Automatic Variables defined outside any function can be seen inside every function.

Static Variables defined outside any function can be seen inside the functions in that file.

/\*

1.10.c - External variables and scope

\*/

#include <stdio.h>

// robert variables are initialized when the program starts:

int robert = 0 ; /\* automatic global: seen everywhere \*/

static int filerobert = 37 ; /\* static global: seen in this file \*/

int func\_one(int a, int b)

{

static int times\_called = 0 ; /\* static : exists between calls to this function \*/

int ghost ; /\* automatic: deleted and re-created each function call \*/

++ robert\_x ;

++times\_called ;

if (times\_called < 2)

printf("in func\_one() seeing robert\_x: %d\n", robert\_x) ;

return times\_called ;

}

int main ()

{

extern int robert\_x ; /\* declares global that may be in another source file \*/

int main\_x = 44 ;

++robert\_x ;

printf("in main() see main\_x: %d robert\_x:%d...\n", main\_x, robert\_x) ;

printf("calling func\_one, getting: %d\n", func\_one(0,0)) ;

printf("calling func\_one, getting: %d\n", func\_one(0,0)) ;

printf("calling func\_one, getting: %d\n", func\_one(0,0)) ;

}

====================================================

A DEEPER DISCUSSION OF THE MATERIAL:

O2Lecture.notes Chapter 2, Storing Data

The data type computers were built to use most is the "int", or integer,

a whole number.

variable declarations and types

aritmmetic operators

if() and decision making

escape characters

cout <<

using namespace std ;

main()

Escape characters

int variables

+ - \* / % (modulo operator)

Order of operations

EVERYthing is a number in the computer:

Characters are small integers, 8 bits long (1 byte).

Characters can be printible, or non-printing.

They are arranged in the ASCII ("ask-ee") Sequence, and come from the

1930's Teletype machines. Thus, the characters 0-31 are "Control"

characters which were sent to make the receiveing machine do things

other than print letters.

Strings are arrays of characters used to store words and sentences.

The big question you must ask before declaring variables is, "Do I intend

to do math with this, or is it just for display?"

If it's for display, use chars and strings, otherwise use ints and

doubles.

Read the chapter's discussion of the #include statement, and the

Conditional Operator.

THE MOMMA ELEPHANT IN THE ROOM: MOST PEOPLE HAVE ONLY USED

WINDOWS, AND DON'T KNOW WHAT COMPUTERS REALLY LOOK LIKE.

The GUI is a crutch. It eats up resources and slows the world down.

Programs are meant to be run from a command line. Even when you point

and click, the Operating System constructs a command to start the program

and provide any additional information.

The thing is, to WRITE GUI programs takes a LOT of resources and

libraries. It stifles innovation and development of small tools

that are useful for doing real work. We sacrifice 'way too much for

a bunch of lazy users who can't learn to type. That won't change,

but real (industrial strength) computing doesn't work that way.

Computer specialists learn how to drive. And fly. And type. Grrrrr.

So when you need GUI, use Visual Basic, C#, or Java. You can use

C++, but it's inefficient.

USING THE CLI (Command Line Interface):

When you write a program, you can click CodeBlocks->Debug to run

it, but you need to understand how to use the Command Line

Interface, too.

If you write a program to count vowels, you can just start it,

then begin typing, it will read what you type at the keyboard.

BUT: what if you have a huge file you want the program to process?

Then start the program with the Input Redirection Operator: <

a.out < hugefile.txt

That makes the program read the file while thinking you're typing in

the input.

Here's an example:

/\*-----------------------------------------------------------------

\* toupper.cpp - what it sounds like

Sun Jan 3 14:00:29 PST 2010

-----------------------------------------------------------------\*/

#include <cstdio>

#include <cctype>

#include <iostream>

#include <iomanip>

using namespace std ;

int main ()

{

int c = cin.get() ;

while (!cin.eof())

{

c = toupper(c) ;

cout.put(c) ;

c = cin.get() ;

}

}

-------------------------------

Since you're not kiddies, here's the way a real programmer

might write it:

(Less is more.)

/\*-----------------------------------------------------------------

\* short\_toupper.cpp - what it sounds like

Sun Jan 3 14:00:29 PST 2010

-----------------------------------------------------------------\*/

#include <cstdio>

#include <cctype>

#include <iostream>

#include <iomanip>

using namespace std ;

int main ()

{

while (!cin.eof()) cout.put(toupper(cin.get())) ;

}

Read and compare them, be sure you understand what's going on.

Now: can you write a program which makes all characters lowercase,

using similar logic?

Now: how would you write a program which capitalizes all Vowels, and

makes all other letters lower case?

There are two functions: toupper() and tolower() you can use. They are

wonderful, you can send any char to them, and if they can't change

its case (like a period or comma) they just return the unchanged char.

Try it. You don't need to use the if(isalpha()) statement, just

toupper or tolower it.

------------

How files are organized on disk:

Most systems like Windows have filesystems that are based on the

Unix system. Windows separates the system into "Volumes" like A:, C:,

F:, etc. They correspond to floppy, hard, and other kinds of drives

such as network-mounted volumes..

(Unix has a single uniform file system, and devices are attached to

it with "mount" and detached with "umount". "mount" even permits a

type called NFS for "Network FileSystem) so you can mount a disk on

another system.

You "log to" a directory with the cd (Change Directory)

command.

You list the contents of a directory with:

Windows: dir

Unix : l (really: ls -l, I've provided you with an alias.

To see your aliases, type alias at the prompt.)

SPACES DELIMIT PARTS OF A COMMAND.

DON'T RUN THE PARTS TOGETHER.

ENTER EXECUTES THE COMMAND.

Other commands:

mkdir rmdir cd l echo chmod chown umask

file type cat tac less list touch lsattr, chattr

Preprocessor directives. How to program the process

of compilation.

#include <filename>

int main () ;

main() is a function.

main() 's first line is where execution begins.

main() returns an integer value

Complete the self-review exercise at the end of Chapter 2.

Do it over until your score is above 90%.

Be prepared to do Exercise 2.23 ("Largest and Smallest") in lab.

Now that everybody's had time to read Chapters 1 and 2, let's make sure

we understand.

Read this:

/\*-----------------------------------------------------------------

\* sample01.cpp - show some program basics.

Mon Aug 27 16:00:21 PDT 2012

-----------------------------------------------------------------\*/

#include <cstdio>

#include <iostream>

#include <iomanip>

using namespace std ;

int main ()

{

int i , j ;

for ( i = 0 ; i < 1000000 ; i++)

{

cout << setfill('0') << setw(6) << i ;

for (j = 0 ; j < 6 ; j++)

cout.put('\b') ;

} // for i ends

cout.put('\n') ;

} // main ends

What do we see?

0. a new #include <iomanip> statement

1. function: int main() ;

2. 2 variables, i and j, both signed integers

3. output cout <<

4. single character output ( putchar() ) // cout.put()

5. nested loops (for each iteration of for i, for j runs 6 times)

6. escape characters (\b backspace, \n newline ).

=========================================================

/\*-----------------------------------------------------------------

\* sample01.cpp - show some program basics.

Mon Aug 27 16:00:21 PDT 2012

-----------------------------------------------------------------\*/

#include <cstdio>

#include <iostream>

#include <iomanip>

using namespace std ;

int main ()

{

int i , j ;

for ( i = 0 ; i < 1000000 ; i++)

{

printf("%06d", i ) ; // cout << setfill('0') << setw(6) << i ;

for (j = 0 ; j < 6 ; j++)

putchar('\b') ; // cout.put('\b') ;

} // for i ends

putchar('\n') ;

} // main ends

/\*-----------------------------------------------------------------

\* sample02.cpp - show some program basics.

Mon Aug 27 16:18:35 PDT 2012

-----------------------------------------------------------------\*/

#include <cstdio>

#include <iostream>

#include <iomanip>

using namespace std ;

int main ()

{

int i , count = 0 , total = 0 ;

double average = 0. ;

cout << "Enter a series of integers, 0 to stop" << endl ;

do

{

cin >> i ;

if (i == 0 )

break ;

total += i ;

count++ ;

printf("%2d TOTAL: %06d\n", count , total ) ; // cout << setfill('0') << setw(6) << i ;

} while (i != 0 ) ;

average = (double) total / (double) count ;

//cout << setprecision(5) << "Average "

// << average << " total: " << total

// << " count: " << count << endl ;

printf("Average: %9.5f total: %4d count: %4d\n", average, total, count) ;

} // main ends

/\*-----------------------------------------------------------------

\* sample03.cpp - a number series (Fibonacci)

\* each number is the sum of the two before it.

Mon Aug 27 16:37:43 PDT 2012

Challenge: eliminate the array "array[]" and still

accomplish the same thing. Think.

-----------------------------------------------------------------\*/

#include <cstdio>

#include <iostream>

#include <iomanip>

using namespace std ;

int main ()

{

int i ;

unsigned int array[80] = {0} ;

getchar() ;

array[0] = 1 ;

array[1] = 2 ;

for (i = 2 ; i < 80 ; i++) {

array[i] = array[i-1] + array[i-2] ;

}

for (i = 0 ; i < 50 ; i++) printf("%12u \n", array[i]) ;

putchar('\n') ;

} // main ends

/\*-----------------------------------------------------------------

\* sample04.cpp - fix this so it displays the answers

\* to the questions it's asking.

Tue Jun 3 23:41:43 PDT 2014

-----------------------------------------------------------------\*/

#include <iostream>

using namespace std ;

int main ()

{

int x, y , i ;

x = y = i = 0 ;

x = y++ ;

// question 1: what are the values of x and of y here?

x = ++y ;

// question 2: what are the values of x and of y here?

i = (x == y) ;

// question 3: what is the value of i here?

}

// question 4: if you modify this program to prove your answers, what

// would it look like?