BLOCK 14.

We're looking at avanced stuff in C++ now, and cleaning up some of the

odds and ends of C. First, we should look at both kinds of goto in C.

Now, I showed you simple goto before we even learned loops, so you

understood how loops work. In the computer's Machine Language,

there is ONLY goto, it's the only way we have of controlling execution.

It's restricted: you can only goto a label: within the same function.

You can't jump from void print\_output() to a line in int main() for

instance.

But there is also setjmp and longjmp. These tools allow us to store

the system's "state", and then to leap back in time to when that

state was stored.

Most "modern" languages frown on goto.

That's fine, like handrails and guard rails are fine. Like Dead-Man's

Switches, Safety Lanyards, engine governors, all the impedimenta of

incompetence and inattention are fine.

But not for you if you really have to make machines work.

Some people work in contexts where safety hardware can kill them.

Sometimes the best thing is for management to cancel the extensible

crain truck, hand the employee a chainsaw, go to the lunch room, and

let someone who knows what they're doing get on with the dirty jobs.

C is a chainsaw. C++ is a CNC milling machine.

But, (it has to be said) MOST of MOST programmer's code is there to

accomodate the ignorance of users.

Imagine that a job must start whether the operator is there to tell

it "daily" or "monthly" processing or not, it MUST begin by a certian

time. (Nightly Backup comes to mind.) The system prompts, and if nobody

answers in a certain number of seconds, it takes the bit in its teeth

and starts a nightly run.

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

\* timeinput.c - do non-blocking input with

\* a time limit using longjmp() .

\* This is useful for situations where console

\* operator may be absent and pgm should continue executing with

\* defaults rather than hang, due to time constraints.

\* This program uses system-level i/o, so if you don't know open(),

\* read(), etc., you 'll need to look them up.

\* TEK verified Thu Jun 22 16:11:18 PDT 2000

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

#include <signal.h>

#include <unistd.h>

#include <setjmp.h>

#include <stdio.h>

#include <fcntl.h>

#include "/usr/local/include/getchne.h"

jmp\_buf context ; /\* type jmp\_buf contains registers \*/

int cc, errno, count = 1 ;

void alarm\_handler(int sig) ;

main()

{

char buffer[81] ;

signal (SIGALRM, alarm\_handler) ;

cc = -1 ;

puts ("N)ightly or W)eekly backup? Pres N or W.") ;

if (setjmp(context)) /\* if returning from a goto... \*/

{

errno = (int) SIG\_ERR ;

puts ("No input within time limit, starting to run NIGHTLY backup.") ;

}

else /\* setjump just initialized \*/

{

alarm(5) ; /\* timeout after 5 seconds \*/

cc = getchne() ;

alarm(0) ; /\* received input, disable alarm \*/

switch (cc)

{

case 'n' :

case 'N' : puts ("NIGHTLY backup starting....") ; break ;

case 'w' :

case 'W' : puts ("WEEKLY backup starting....") ; break ;

default : puts ("NIGHTLY backup starting....") ; break ;

}

}

} /\*main ends \*/

void alarm\_handler(int sig) {

printf("at alarm\_handler with sig of: %d\n", sig) ;

if (cc == -1) longjmp(context, 1) ; else return ;

}

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

Speaking of time, here's an example of how to get local time

and display it:

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

\* timehack.c from tbC BBS, 1993.

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

#include <stdio.h>

#include <time.h>

#include <string.h>

char \* timehack(void)

{

struct tm \* tm1 ;

time\_t time\_t\_1 ;

char timebuff[81] ;

time(&time\_t\_1) ; // library fcn allocates and fills buffer time\_t\_1

tm1=localtime(&time\_t\_1) ; // library fcn allocates and fills buffer tm1

strncpy(timebuff, asctime(tm1), sizeof(timebuff)) ; // format output

timebuff[24] = '\0' ; // remove newline

return timebuff ;

} /\* timehack ends \*/

int main()

{

char buff[81] ;

strncpy(buff, timehack(), 81) ;

printf("%s\n", buff) ;

} // main ends

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

At the beginning of the course, I said that if you learn C, you will

know most of a handful of other programming languages.

As an example, here is grade.awk. I wrote it to grade the midterm

answer files. It compares the "h" answer key with the "maxNN"

test file. Awk uses C syntax, but automatically opens and

processes each line in a file without the programmer needing to

code an input loop. For daily processing awk is a time saver.

A more refined version of awk is PERL ("Practial Extraction and

Reporting Language"). It moves most of the information on the

internet, and most servlets are written in it.

/usr/bin/awk '

{

# run this: paste h max27 | sed 's/^M//g' | ./grade.awk | less

# Fields: $1: answer key $2 line no in key

# $3: line no in test $4: answer in test

$1 = tolower($1) ;

$4 = tolower($4) ;

if ($4 == "f") $4 = "b" ;

if ($4 == "t") $4 = "a" ;

printf("%s %s %s %s", $1, $2, $3, $4) ;

if (length($4)==1)

{

++count ;

if (tolower($1) != tolower($4))

{

printf(" ---") ; #mark error

wrong++ ;

}

}

printf("\n") ;

}

END {

printf("%d wrong\n", wrong) ;

right = count - wrong ;

printf("%d / %d\n", right , count ) ;

printf("Score: %d\n", right \* 50 / 40 ) ;

}

' $\*

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

The ability to move around in files keeps every storage device from

performing like a slow tape drive where, if you want record 10,

you must first read 1 through 9. With seek commands, you may

go to the data you need efficiently.

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

\* randomaccess.cpp - how to control the file pointer

\*

All C files are "random access" because all C file i/o is binary,

even if the bytes are automatically treated as chars.

print argv[1] copies of file, then

turn file inside-out (print a disk file backwards)

To do this with a dynamic stream, fseek won't work, so take all

the input into a string object and process it.

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

#include <cstdio>

#include <cctype>

#include <iostream>

#include <iomanip>

#include <cstdlib>

using namespace std ;

int main (int argc, char \*argv[], char \*\*env)

{

FILE \* inp ;

int c ,a ;

long filelength ;

//int reps = atoi(argv[1]) ;

inp = fopen(argv[1], "rb") ;

if (inp == 0)

{

fprintf(stderr, "Cannot open file: %s ending run\n", argv[1]) ;

return -1 ;

}

c = fgetc(inp) ;

while (!feof(inp))

{

putchar(c) ;

c = fgetc(inp) ;

}

printf("===================================\n") ;

fseek(inp, -1L, SEEK\_END) ; // avoid EOF mark:

while (a != 0 )

{

c = fgetc(inp) ;

putchar(c) ;

fseek(inp, -2L , SEEK\_CUR) ; // go back 1 char before last printed

a = ftell(inp) ;

}

fclose(inp) ;

putchar('\n') ;

} // main ends

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

Structures and Unions:

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

Structs, simpler precursors to classes:

A struct is a way of giving a name to a block of

memory which you're going to use to store some

dissimilar (more than one type, int and double,

or char and bool and long) data items in. We use

them a lot to create databases and for other kinds

of business data processing. Structs were the last

development before Objects were developed, and Objects

and Structs are similar.

This chapter goes a bit overboard, like on P.116

where it tries to store a function inside a struct.

Don't waste too much time on this (P.116-119) because

it is showing struct usage (public:/private:) that is

ONLY AVAILABLE in C++, not C, and if you're using

C++, you'd use CLASSES in preference to structs.

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

A union is a way of looking at the same memory

more than one way. Consider:

union ut

{

int i ;

char c[4] ;

} U ;

U is the name for a union which can contain either a

signed int or an array of four bytes (char). You can

load a number into the integer, then print out

each char, and see how the computer stores the int,

byte by byte. Unions are mainly used when we're

controlling devices at a low level, and by students

interested in seeing how the computer works.

U.i = -21238 ;

cout << (int) U.c[0] << endl ;

cout << (int) U.c[1] << endl ;

cout << (int) U.c[2] << endl ;

cout << (int) U.c[3] << endl ;

In over 20 years of programming, I've never really

needed to use a union in an application program.

They were a lot more useful back when computer memory

was at a premium, or when writing hardware-specific

code such as device drivers.

They're never needed in a language that is "weakly typed"

like C. But they permit clearer code, and are a way

of taking apart multi-byte data to see how it's really stored.

Remember them in your Computer Hardware class.

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

Bit Fields

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

Boolean operators.

AND OR XOR NOT

& | ^ ~

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

Read: <A HREF="http://209.129.16.61/~hhaller/data/cisc192/modules/bittrans.c.txt">bittrans.c</A>

Read: <A HREF="http://209.129.16.61/~hhaller/data/cisc192/modules/tstbit.c.txt">tstbit.c</A>

C++ Bit Fields: these are not as useful as they should have been, since

you can't access them with for() loops. But they can be used to map

devices for easy programmer access. If you can allocate an array of

unsigned char and manipulate individual bits in the array using / and %,

that is the main skill you must have, everything else is just convenience.

C++ provides the BITSET, which is slightly more useful, but still not

what it should have been.

To learn how to use C to the optimum at this level, you should practice

programming in C for a few months, then get a good Assembly Language

textbook and read the sections on data transformations. Most of the

mid-level C programs I wrote, were inspired by "Using Assembly Language"

by Wyatt. C is ideally suited to assembler-level tasks, but always

remember that the programs won't be portable if the hardware they run

on has a different processor, or is Little-Endian when you wrote if for

Big-Endian, or vice versa.

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

\* randomaccess.cpp - how to control the file pointer

\*

All C files are "random access" because all C file i/o is binary,

even if the bytes are automatically treated as chars.

print argv[1] copies of file, then

turn file inside-out (print a disk file backwards)

To do this with a dynamic stream, fseek won't work, so take all

the input into a string object and process it.

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

#include <cstdio>

#include <cctype>

#include <iostream>

#include <iomanip>

#include <cstdlib>

using namespace std ;

int main (int argc, char \*argv[], char \*\*env)

{

FILE \* inp ;

int c ,a ;

long filelength ;

//int reps = atoi(argv[1]) ;

inp = fopen(argv[1], "rb") ;

if (inp == 0)

{

fprintf(stderr, "Cannot open file: %s ending run\n", argv[1]) ;

return -1 ;

}

c = fgetc(inp) ;

while (!feof(inp))

{

putchar(c) ;

c = fgetc(inp) ;

}

printf("===================================\n") ;

fseek(inp, -1L, SEEK\_END) ; // avoid EOF mark:

while (a != 0 )

{

c = fgetc(inp) ;

putchar(c) ;

fseek(inp, -2L , SEEK\_CUR) ; // go back 1 char before last printed

a = ftell(inp) ;

}

fclose(inp) ;

putchar('\n') ;

} // main ends

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

\* timeinput.c - do non-blocking input with

\* a time limit using longjmp() .

\* This is useful for situations where console

\* operator may be absent and pgm should continue executing with

\* defaults rather than hang, due to time constraints.

\* This program uses system-level i/o, so if you don't know open(),

\* read(), etc., you 'll need to look them up.

\* TEK verified Thu Jun 22 16:11:18 PDT 2000

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

#include <signal.h>

#include <unistd.h>

#include <setjmp.h>

#include <stdio.h>

#include <fcntl.h>

#include "/usr/local/include/getchne.h"

jmp\_buf context ; /\* type jmp\_buf contains registers \*/

int cc, errno, count = 1 ;

void alarm\_handler(int sig) ;

main()

{

char buffer[81] ;

signal (SIGALRM, alarm\_handler) ;

cc = -1 ;

puts ("N)ightly or W)eekly backup? Pres N or W.") ;

if (setjmp(context)) /\* if returning from a goto... \*/

{

errno = (int) SIG\_ERR ;

puts ("No input within time limit, starting to run NIGHTLY backup.") ;

}

else /\* setjump just initialized \*/

{

alarm(5) ; /\* timeout after 5 seconds \*/

cc = getchne() ;

alarm(0) ; /\* received input, disable alarm \*/

switch (cc)

{

case 'n' :

case 'N' : puts ("NIGHTLY backup starting....") ; break ;

case 'w' :

case 'W' : puts ("WEEKLY backup starting....") ; break ;

default : puts ("NIGHTLY backup starting....") ; break ;

}

}

} /\*main ends \*/

void alarm\_handler(int sig) {

printf("at alarm\_handler with sig of: %d\n", sig) ;

if (cc == -1) longjmp(context, 1) ; else return ;

}