**18** System Programming Projects

Reading a Binary File in C on a Windows System

FILE \*infile, \*outfile;

infile = fopen("x.in", "r");

outfile = fopen("x.out", "w");

infile = fopen("x.in", "rb"); // "b" qualifier needed on Windows

outfile = fopen("x.out", "wb");

Reading and Writing Binary Files

char buf[100];

int count;

count = fread(buf, 1, sizeof(buf), infile);

Address of input area

Max number of units to read

count = fread(&x, sizeof(int), 1, infile);

Reading char

c = fgetc(infile);

if (c == EOF)

{

...

}

signed char c; // sign extended

unsigned char c; // zero extended  
 char c; // extension is compiler dependent

Write to a Binary File

count = fwrite(buf, sizeof(buf), 1, outfile);

fputc(c, outfile);

Input and Output Functions in C for Text Files

count = fscanf(infile, "%d", &x);

fprintf(outfile, "x = %d\n", x);

fgets(buf, sizeof(buf), infile);

while(fgets(buf, sizeof(buf), infile))  
 {

}

fputs(buf, outfile);

fputs("hello\n", outfile);

Little Endian and Big Endian

Little Endian Big Endian

register

register

12345678

memory

memory

12345678

~ ~ ~ ~

78 1001 12 1001

56 1002 34 1002

34 1003 56 1003

12 1004 78 1004

~ ~ ~ ~

Using Masks and Bitwise Operators

y = x & 0x1ff;

00...01010001 000010010 x

00...00000000 111111111 mask

00...00000000 000010010 result

if (x & 0x20)

{

*execute if sixth bit from right is 1*

}

00...0100000

y = x | 0x20;

00...010100010 00010010 x

00...000000000 00100000 mask

00...010100010 00110010 result

Accessing Command Line Arguments

test hello goodbye

#include <stdio.h>

int main(int argc, char \*argv[])  
{  
 int i;  
 for (i = 0, i < argc; i++)  
 printf("%s\n", argv[i]);

return 0;

}

test hello goodbye

argc argv

"C:\a\cuh\test.exe"

3

"hello"

"goodbye"

C:\a\cuh\test.exe

hello

goodbye

test hello goodbye > test.out

test.out

C:\a\cuh\test.exe

hello

goodbye

Common Bugs in C Programs

int \*p; // bad  
 \*p = 7;

int x, \*p; // good

p = &x; // assign p the address of x  
 \*p = 7; // ok to dereference p

p = (int \*)malloc(sizeof(int)); // assign p addr of allocated storage  
 \*p = 7; // ok to dereference p

char a[100]; // ugly

a = "hello";

strcpy(a, "hello");

char \*p = "hello";

char \*q = "bye";

strcat(p, q); // do not do this!

char buf[100];

strcpy(buf, p); // copy "hello" into buf

strcat(buf, q); // concatenate "bye" to "hello" in buf  
 p = buf; // assign p the address of concatenated string

printf("%s\n", buf); // displays hellobye

printf("%s\n", p); // displays hellobye

printf("x = ", x); // wrong: needs one conversion code in 1st arg

printf("x = %d\n", x); // right: one conversion code in 1st arg

Field Width in printf Conversion Codes

printf("---%d---\n", 123);

displays

---123---

printf("---%8d---\n", 123);

displays

8 is the field width

--- 123---

printf("---%4d---\n", 12345678);

displays

---12345678---

printf("---%08d---\n", 123);

displays

---00000123---

Displaying a Byte in Hex

unsigned char uc;

signed char sc;

char c;

* i and j have the type int and are 32-bits wide
* the eight-bit value in both uc and sc is F1 hex (11110001 in binary)

i = uc; // i is assigned 241 (000000F1 hex)

j = sc; // j is assigned -15 (FFFFFFF1 hex)

*Rule*: The type of extension—zero or signed—*depends on the type of the variable* extended.

printf("%02X", uc); // displays F1

printf("%02X", sc); // displays FFFFFFF1

printf("%02X", sc & 0xff); // displays F1

printf("%02X", sc); // displays FFFFFFF1 Why???

printf("%c", sc); // displays what???

The calling sequence extends the value in sc to 32 bits before passing it to printf.

Because of the conversion code is %c, printf accesses and displays the character *in only the low-order byte* of the 32-bit value it is passed.

Hex/ASCII Display Project

h1 h1test.txt

displays

YOUR NAME HERE h1 h1test.txt Thu Jun 03 16:50:08 2021

0: 5468 6973 2070 726F 6772 616D 2061 6C6C

10: 6F77 7320 796F 7520 746F 206C 6F6F 6B20

20: 696E 7369 6465 2061 2066 696C 6520 746F

30: 2073 6565 2077 6861 7420 6973 200D 0A72

40: 6561 6C6C 7920 7468 6572 652E 2042 7574

50: 2074 6F20 6265 2075 7365 6675 6C2C 2079

60: 6F75 206E 6565 6420 746F 206B 6E6F 7720

70: 6865 7861 6465 6369 6D61 6C20 0D0A 6E6F

80: 7461 7469 6F6E 2E20 5265 6D65 6D62 6572

90: 2061 203D 2031 302C 2062 203D 2031 312C

A0: 2063 203D 2031 322C 2064 203D 2031 332C

B0: 2065 203D 2031 342C 200D 0A61 6E64 2066

C0: 203D 2031 352E 0D0A 0001 FFF3 4279 65

// h1shell.c

*Your name here as a comment*

#include <stdio.h> // for I/O

#include <stdlib.h> // for exit()

#include <time.h> // for time functions

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

{

FILE \*infile;

int i, numread;

unsigned char buf[32768];

time\_t timer;

if (argc != 2)

{

printf("Wrong number of command line arguments\n");

printf("Usage: h1 <inputfilename>\n");

exit(1);

}

// display your name, command line args, and time

Replace with your name

time(&timer);

printf("YOUR NAME HERE %s %s %s",

argv[0], argv[1], asctime(localtime(&timer)));

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

if (!infile)

{

printf("Cannot open input file %s\n", argv[1]);

exit(1);

}

numread = fread(buf, 1, sizeof(buf), infile);

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

{

*Code missing here:*

*Display buf[i] so that 16 bytes appear on each line,*

*with a space between each pair of bytes as shown in the textbook.*

*Use i to determine when to insert space and newline. Start each*

*line with the hex address of the start of that line followed by a colon.*

}

}

No Limit on File Size for h2

// h2shell.c

*Your name here as a comment*

#include <stdio.h> // for I/O

#include <stdlib.h> // for exit()

#include <time.h> // for time functions

FILE \*infile;

// nextbyte() handles any input file size

int nextbyte()

{

*Declaring the following variables static so they retain their values between calls.*

static unsigned char buf[100];

static int numread, bufindex = sizeof(buf);

if (bufindex == sizeof(buf))

{

*Code missing here:*

*Read next 100-byte block and reset bufindex to 0.*

*Assign to numread the number of bytes actually read.*

}

if (bufindex < numread)

{

*Code missing here:*

*Return byte in buf[bufindex] as an int.*

*Increment bufindex.*

}

else

return -1; // -1 signals end of file

}

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

{

int i, byte;

time\_t timer;

if (argc != 2)

{

printf("Wrong number of command line arguments\n");

printf("Usage: h2 <inputfilename>\n");

exit(1);

}

// display your name, command line args, and time

Replace with your name

time(&timer);

printf("YOUR NAME HERE %s %s %s",

argv[0], argv[1], asctime(localtime(&timer)));

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

if (!infile)

{

printf("Cannot open input file %s\n", argv[1]);

exit(1);

}

i = 0; // use i to determine when to insert space and newline

while (1)

{

byte = nextbyte();

if (byte == -1)

break;

*Code missing here:*

*Display byte here so that 16 bytes appear on each line,*

*with a space between each pair of bytes as shown in the textbook.*

*Use i to determine when to insert space and newline. Start each*

*line with the hex address of the start of that line followed by a colon.*

i++;

}

}

h3 Displays Characters Corresponding to ASCII Codes

h3 h3test.txt > h3test.out

YOUR NAME HERE h3 h3test.txt Thu Jun 03 16:50:08 2021

0: 5468 6973 2070 726F 6772 616D 2061 6C6C This program all

10: 6F77 7320 796F 7520 746F 206C 6F6F 6B20 ows you to look

20: 696E 7369 6465 2061 2066 696C 6520 746F inside a file to

30: 2073 6565 2077 6861 7420 6973 200D 0A72 see what is ..r

40: 6561 6C6C 7920 7468 6572 652E 2042 7574 eally there. But

50: 2074 6F20 6265 2075 7365 6675 6C2C 2079 to be useful, y

60: 6F75 206E 6565 6420 746F 206B 6E6F 7720 ou need to know

70: 6865 7861 6465 6369 6D61 6C20 0D0A 6E6F hexadecimal ..no

80: 7461 7469 6F6E 2E20 5265 6D65 6D62 6572 tation. Remember

90: 2061 203D 2031 302C 2062 203D 2031 312C a = 10, b = 11,

A0: 2063 203D 2031 322C 2064 203D 2031 332C c = 12, d = 13,

B0: 2065 203D 2031 342C 200D 0A61 6E64 2066 e = 14, ..and f

C0: 203D 2031 352E 0D0A 0001 FFF3 4279 65 = 15.......Bye

Machine Interpreter Level 1 Project

// i1shell.c

*Your name here as a comment*

#include <stdio.h> // for I/O functions

#include <stdlib.h> // for exit()

FILE \*infile;

short r[8], mem[65536], offset6, imm5, imm9, pcoffset9, pcoffset11,

regsave1, regsave2;

unsigned short ir, pc, opcode, code, dr, sr, sr1, sr2, bit5, bit11,

trapvect8, n, z, c, v;

char letter;

time\_t timer;

void setnz(short r)

{  
 n = z = 0;

if (r < 0) // is result negative?

n = 1; // set n flag

else

if (r == 0) // is result zero?

z = 1; // set z flag

}

void setcv(short sum, short x, short y)

{

v = c = 0;

if (x >= 0 && y >= 0) // if both non-negative, then no carry

c = 0;

else

if (x < 0 && y < 0) // if both negative, then carry

c = 1;

else

if (sum >= 0) // if signs differ and sum non-neg, then carry

c = 1;

else // if signs differ and sum neg, then no carry

c = 0;

// if signs differ then no overflow

if ((x < 0 && y >= 0) || (x >= 0 && y < 0))

v = 0;

else

// if signs the same and sum has different sign, then overflow

if ((sum < 0 && x >= 0) || (sum >= 0 && x < 0))

v = 1;

else

v = 0;

}

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

{

printf("Wrong number of command line arguments\n");

printf("Usage: i1 <inputfilename>\n");

exit(1);

}

// display your name, command line args, time

time(&timer); // get time

printf("YOUR NAME HERE %s %s %s",

argv[0], argv[1], asctime(localtime(&timer)));

infile = fopen(argv[1], "rb"); // open file in binary mode

if (!infile)

{

printf("Cannot open input file %s\n", argv[1]);

exit(1);

}

fread(&letter, 1, 1, infile); // test for and discard file sig

if (letter != 'o')  
 {

printf("%s not an lcc file\n", argv[1]);

exit(1);

}

fread(&letter, 1, 1, infile); // test for and discard 'C'

if (letter != 'C')  
 {

printf(("Missing C header entry in %s\n", argv[1]);

exit(1);

}

fread(mem, 1, sizeof(mem), infile); // read machine code into mem

while (1)

{

// fetch instruction, load it into ir, and increment pc

ir = mem[pc++];

// isolate the fields of the instruction in the ir

opcode = ir >> 12; // get opcode

pcoffset9 = ir << 7; // left justify pcoffset9 field

pcoffset9 = imm9 = pcoffset9 >> 7; // sign extend and rt justify

pcoffset11 = ... // left justify pcoffset11 field

pcoffset11 = ... // sign extend and rt justify

imm5 = ... // left justify imm5 field

imm5 = ... // sign extend and rt justify

offset6 = ... // left justify offset6 field

offset6 = ... // sign extend and rt justify

trapvect8 = ir & 0xff; // get trapvect8 field

code = dr = sr = ... // get code/dr/sr, rt justify

sr1 = baser = (ir & 0x01c0) >> 6; // get second reg, rt justify

sr2 = ... // get third reg

bit5 = ... // get bit 5

bit11 = ir & 0x0800; // get bit 11

// determine and execute instruction just fetched

switch (opcode)  
 {

case 0: // branch instructions

switch(code)  
 {  
 case 0: if (z == 1) // brz

pc = pc + pcoffset9;  
 break;  
 case 1: if (z == 0) // brnz

pc = pc + pcoffset9;  
 break;

*code missing here*

case 7: pc = pc + pcoffset9; // br

break;  
 }

break;

case 1: // add

if (bit5)

{

regsave1 = r[sr1];

r[dr] = regsave1 + imm5;

// set c, v flags

setcv(r[dr], regsave1, imm5);

}

else

{

regsave1 = r[sr1]; regsave2 = r[sr2];

r[dr] = regsave1 + regsave2;

// set c, v flags

setcv(r[dr], regsave1, regsave2);

}

// set n, z flags

setnz(r[dr]);

break;

case 2: // ld

*code missing here*

case 9: // not

// ~ is the not operator in C

r[dr] = ~r[sr1];

// set n, z flags

setnz(r[dr]);

break;

*code missing here*

case 12: // jmp/ret

pc = r[baser];

break;

*code missing here*

case 14: // lea

r[dr] = pc + pcoffset9;

break;

case 15: // trap

if (trapvect8 == 0x00) // halt

exit(0);

else  
 if (trapvect8 == 0x01) // nl  
 *code missing here*

else

if (trapvect8 = 0x02) // dout  
 *code missing here*

break;

}

}

Assembler Level 1 Project

// a1shell.c

*Your name here as a comment*

#include <stdio.h> // for I/O functions

#include <stdlib.h> // for exit()

#include <string.h> // for string functions

#include <ctype.h> // for isspace(), tolower()

#include <time.h> // for time functions

FILE \*infile, \*outfile;

short pcoffset9, pcoffset11, imm5, imm9, offset6;

unsigned short symadd[500], macword, dr, sr, sr1, sr2, baser, trapvect8;

char outfilename[100], linesave[100], buf[100], \*symbol[500], \*p1, \*p2,

\*mnemonic, \*o1, \*o2, \*o3, \*label;

int stsize, num, linenum, rc, loc\_ctr;

time\_t timer;

// case insensitive string compare

short int strcmpi(const char \*p, const char \*q)

{

// Returns 0 if two strings are equal.

char a, b;

while (1)

{

a = tolower(\*p); b = tolower(\*q);

if (a != b) return a-b;

if (a == '\0') return 0;

p++; q++;

}

}

void error(char \*p)

{

*Code missing here:*

*Displays error message p points to, line number in linenum, and line in linesave.*

}

int isreg(char \*p)

{

*Code missing here:*

*Returns 1 if p points to a register name. Otherwise, returns 0.*

}

unsigned short getreg(char \*p)

{

*Code missing here:*

*Returns register number of the register whose name p points to.*

*If p does not point to a register name, call error().*

}

unsigned short getadd(char \*p)

{

*Code missing here:*

*Returns address of symbol that p points by accessing the symbol table.*

*Calls error() if symbol not in symbol table.*

}

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

{

if (argc != 2)

{

printf("Wrong number of command line arguments\n");

printf("Usage: a1 <inputfilename>\n");

exit(1);

}

// display your name, command line args, and time

Replace with your name

time(&timer);

printf("YOUR NAME HERE %s %s %s",

argv[0], argv[1], asctime(localtime(&timer)));

infile = fopen(argv[1], "r");

if (!infile)

{

printf("Cannot open input file %s\n", argv[1]);

exit(1);

}

// construct output file name

strcpy(outfilename, argv[1]); // copy input file name

p1 = strrchr(outfilename, '.'); // search for period in extension

if (p1) // name has period

{

#ifdef \_WIN32 // defined only on Windows systems

p2 = strrchr(outfilename, '\\' ); // compiled if \_WIN32 is defined

#else

p2 = strrchr(outfilename, '/'); // compiled if \_WIN32 not defined

#endif

if (!p2 || p2 < p1) // input file name has extension?

\*p1 = '\0'; // null out extension

}

strcat(outfilename, ".e"); // append ".e" extension

outfile = fopen(outfilename, "wb");

if (!outfile)

{

printf("Cannot open output file %s\n", outfilename);

exit(1);

}

loc\_ctr = linenum = 0; // initialize

fwrite("oC", 2, 1, outfile); // output empty header

// Pass 1

printf("Starting Pass 1\n");

while (fgets(buf, sizeof(buf), infile))

{

linenum++; // update line number

p = buf;

while (isspace(\*p)) p++;

if (\*p == '\0' || \*p ==';') // if line all blank, go to next line

continue;

strcpy(linesave, buf); // save line for error messages

if (!isspace(buf[0])) // line starts with label

{

label = strdup(strtok(buf, " \r\n\t:"));

*Add code here that checks for a duplicate label, use strcmp().*

symbol[stsize] = label;

symadd[stsize++] = loc\_ctr;

mnemonic = strtok(NULL," \r\n\t"); // get ptr to mnemonic/directive

o1 = strtok(NULL, " \r\n\t"); // get ptr to first operand

}

else // tokenize line with no label

{

mnemonic = strtok(buf, " \r\n\t"); // get ptr to mnemonic

o1 = strtok(NULL, " \r\n\t"); // get ptr to first operand  
 }

if (mnemonic = NULL) // check for mnemonic or directive  
 continue;

if (!strcmp(mnemonic, ".blkw"))

{

if (o1)  
 rc = sscanf(o1, "%d", &num); // get size of block from o1

else  
 error("Missing operand");

if (rc != 1 || num > (65536 – loc\_ctr) || num < 1)  
 error("Invalid operand");

loc\_ctr = loc\_ctr + num;

}

else

loc\_ctr++;

if (loc\_ctr > 65536)

error("Program too big");

}

rewind(infile);

// Pass 2

printf("Starting Pass 2\n");

loc\_ctr = linenum = 0;

while (fgets(buf, sizeof(buf), infile))  
 {

linenum++;

*Code missing here:*

*Discard blank/comment lines.  
 Save buf in linesave as in pass 1.*

*Tokenize entire current line.*

*Do not make any new entries into the symbol table.*

if (mnemonic == NULL)  
 continue;

if (!strncmp(mnemonic, "br", 2)  
 {

if (!mstrcmpi(mnemonic, "br" ))

macword = 0x0e00;

else

if (!mstrcmpi(mnemonic, "brz" ))

macword = 0x0000;

else

if (!mstrcmpi(mnemonic, "brnz" ))

macword = 0x0200;

else

if (!mstrcmpi(mnemonic, "brn" ))

macword = 0x0400;

else

if (!mstrcmpi(mnemonic, "brp" ))

macword = 0x0600;

else

if (!mstrcmpi(mnemonic, "brlt" ))

macword = 0x0800;

else

if (!mstrcmpi(mnemonic, "brgt" ))

macword = 0x0a00;

else

if (!mstrcmpi(mnemonic, "brc" ))

macword = 0x0c00;

else

error("Invalid branch mnemonic");

pcoffset9 = (getadd(o1) - loc\_ctr - 1); // compute pcoffset9

if (pcoffset9 > 255 || pcoffset9 < -256)

error("pcoffset9 out of range");

macword = macword | (pcoffset9 & 0x01ff); // assemble inst

fwrite(&macword, 2, 1, outfile); // write out instruction

loc\_ctr++;

}

else

if (!strcmp(mnemonic, "add" ))

{

if (!o3)  
 error("Missing operand");

dr = getreg(o1) << 9; // get and position destination reg number

sr1 = getreg(o2) << 6; // get and position source reg 1 number

if (isreg(o3)) // 3rd operand a register?

{

sr2 = getreg(o3);// get third reg number

macword = 0x1000 | dr | sr1 | sr2; // assemble inst

}

else

{

if (sscanf(o3,"%d", &num) != 1) // convert imm5 field  
 error("Bad imm5");

if (num > 15 || num < -16)

error("imm5 out of range");

macword = 0x1000 | dr | sr1 | 0x0020 | (num & 0x1f);

}

fwrite(&macword, 2, 1, outfile); // write out instruction

loc\_ctr++;

}

else

if (!strcmp(mnemonic, "ld" ))

{

dr = getreg(o1) << 9; // get and position destination reg number

pcoffset9 = (getadd(o2) – loc\_ctr - 1);  
 if (pcoffset9 > 255 || pcoffset9 < -256)  
 error("pcoffset9 out of range");  
 macword = 0x2000 | dr | (pcoffset9 & 0x1ff); // assemble inst

fwrite(&macword, 2, 1, outfile); // write out instruction

loc\_ctr++;

}

*code missing here*

else  
 if (!strcmp(mnemonic, "jmp" ))

{

baser = getreg(o1) << 6; // get and position reg number  
 macword = 0xc000 | baser; // assemble instruction

fwrite(&macword, 2, 1, outfile); // write out instruction

loc\_ctr++;

}

*code missing here*

else  
 if (!strcmp(mnemonic, ".zero"))

{

macword = 0;

sscanf(o1, "%d", &num); // get size of block

loc\_ctr = loc\_ctr + num; // adjust loc\_ctr

while (num--) // write out a block of zeros

fwrite(&macword, 2, 1, outfile);

}

else

error("Invalid mnemonic or directive");

}  
 *Close files.*

}

Machine Interpreter Level 2 Project

Extend your i1 interpreter so that it supports, A, S, and C header entries and a command-line-specified load point.

i2 i2test.e 500

To read S and A addresses:

fread(&start, 2, 1, infile); // to read addresses in S and A entries

where start is declared as an unsigned short integer.

Assembler Level 2 Project

fwrite("S", 1, 1, outfile); // output S entry

fwrite(&addr, 2, 1, outfile);

; a2test.a

.start s

halt

s: lea r0, c7

st r0, ac7

ld r0, x

dout ; 1

nl

add r0, r0, r0

dout ; 2

nl

add r0, r0, 1

dout ; 3

nl

add r0, r0, 3

and r0, r0, 4

dout ; 4

nl

br L1

halt

L1:

brp L2

halt

L2:

ABC:

add r1, r0, 0

not r1, r1

brn L3

halt

L3: br L4

X: .word 1

L4: and r2, r2, 0

brz L5

halt

.blkw 3

L5: lea r3, L6

jmp r3

halt

L6: bl r1sub ; 5

lea r4, r2sub

blr r4 ; 6

ld r0, ac7

ldr r0, r0, 0

dout ; 7

nl

lea r4, c6

ldr r0, r4, 4

dout ; 8

nl

ld r5, minus

not r0, r5

dout ; -9

nl

add r0, r0, -1

st r0, save

ld r0, save

dout ; -10

nl

add r0, r0, -1

ld r1, ac7

str r0, r1, 0

ld r0, c7

dout ; -11

nl

add r0, r0, -1

lea r1, save

str r0, r1, 1

ld r0, save2

dout ; -12

nl

halt

; hello

r1sub: ld r0, c5

dout ; 5

nl

ret

c5: .word 5

minus: .word 8

r2sub: ld r0, c6

dout

nl

ret

c6: .word 6

c7: .word 7

ac6: .word c6

ac7: .word c7

c8: .word 8

save: .word -5

save2: .word 100

Module Picture Project

; ptest.a

.global a

.global b

.extern sub

.extern x

.start s

S: bl sub

ld r0,x

dout

halt

a: .word b

b: .word 100

c: .word x

p ptest.o

displays

YOUR NAME HERE p ptest.o Thu Jun 03 16:50:08 2021

Header:

o

S 0000

E 0000 sub

e 0001 x

G 0004 a

A 0004

G 0005 b

V 0006 x

address of b instruction

ld instruction

C

Code:

0: 4800 2000 f002 f000 0005 0064 0000

external reference to x instruction

bl instruction

halt instruction

h3 display:

null-terminated string “sub”

file signature

start of E entry

address 0001 in e entry (in little endian format)

0: 6F53 0000 4500 0073 7562 0065 0100 7800 oS..E..sub.e..x.

10: 4704 0061 0041 0400 4705 0062 0056 0600 G..a.A..G..b.V..

20: 7800 4300 4800 2002 F000 F005 0064 0000 x.C.H. ......d..

30: 00 .

bl instruction (in little endian format)

// pshell.c

*Your name here as a comment.*

#include <stdio.h> // for I/O functions

#include <stdlib.h> // for exit()

#include <time.h> // for time functions

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

{

FILE \*infile;

unsigned short addr, codeword;

char c;

int i;

time\_t timer;

if (argc != 2)

{

printf("Wrong number of command line arguments\n");

printf("Usage: p <inputfilename>\n");

exit(1);

}

infile = fopen(argv[1], “r”);

if (!infile)

{

printf("Cannot open input file %s\n", argv[1]);

exit(1);

}

// display your name, command line args, and time

Replace with your name

time(&timer);

printf("YOUR NAME HERE %s %s %s",

argv[0], argv[1], asctime(localtime(&timer)));

// process header entries

printf("\nHeader:\n");  
 c = fgetc(infile);

if (c == 'o')

printf(" o\n");

else

{

printf("%s not an lcc file\n", argv[1]);  
 exit(1);

}

while (1)

{

c = fgetc(infile);

if (c == 'C')

{

printf(" C\n");

break;

}

if (c == 'S')

{

if (fread(&addr, 2, 1, infile) != 1)

{

printf("Invalid S entry\n");  
 exit(1);  
 }

printf(" S %04hx\n", addr); // %hx conversion code for short

}

else

if (c == 'G')

{

if (fread(&addr, 2, 1, infile) != 1)

{

printf("Invalid G entry\n");  
 exit(1);  
 }

printf(" G %04hx ", addr);

while (1) // read and display string in G entry char by char

{

c = fgetc(infile);

if (c == '\0')

break;

printf("%c", c); // displays characters as read in

}

printf("\n");

}

else

if (c == 'E')

{

*code missing here*

}

else

if (c == 'e')

{

*code missing here*

}

else

if (c == 'V')

{

*code missing here*

}

else

if (c == 'A')

{

*code missing here*

}

else

{

printf("Invalid header entry in %s\n", argv[1]);

exit(1);

}

}

// process machine code

i = 0;

printf("\nCode:\n");

while (fread(&codeword, 2, 1, infile))

{

*Code missing here:*

*Display word in codeword, 8 words per line, 1 space separating*

*each code word, with each line starting with the hex address of*

*the first word on that line as shown in the textbook.*

*Use i to determine when to output hex address and newline.*

i++;

}

}

Linker Project

// lshell.c

*Your name here as a comment*

#include <stdio.h> // for I/O functions

#include <stdlib.h> // for exit()

#include <time.h> // for time functions

int i;

unsigned short temp, inst, addr;

char buf[300];

FILE \*infile;

FILE \*outfile;

char c, \*p, letter;

unsigned short mca[65536];

int mcaindex;

unsigned short start;

int gotstart;

unsigned short Gadd[1000];

char \*Gptr[1000];

int Gindex;

unsigned short Eadd[1000];

char \*Eptr[1000];

int Eindex;

unsigned short eadd[1000];

char \* eptr[1000];

int eindex;

unsigned short Aadd[1000];

int Amodadd[1000];

int aindex;

unsigned short Vadd[1000];

char \*Vptr[1000];

int Vindex;

time\_t timer;

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

{

if (argc < 2)

{

printf("Wrong number of command line arguments\n");

printf("Usage: l <objmodulename1> <objmodulename2> ... \n");

exit(1);

}

// display your name, command line args, time

time(&timer); // get time

printf("YOUR NAME HERE %s %s %s",

argv[0], argv[1], asctime(localtime(&timer)));

//================================================================

// Step 1:

// For each module, store header entries into tables with adjusted

// addresses and store machine code in mca (the machine code array).

for (i = 1; i < argc; i++)

{

infile = fopen(argv[i], "rb");

if (!infile)

{

printf("Cannot open %s\n", argv[i]);

exit(1);

}

printf("Linking %s\n", argv[i]);

letter = fgetc(infile);

if (letter != 'o')

{

printf("Not a linkable file\n");

exit(1);

}

while (1)

{

letter = fgetc(infile);

if (letter == 'C')

break;

else

if (letter == 'S')

{

if (fread(&addr, 2, 1, infile) != 1) // addr unsigned short

{

printf("Invalid S entry\n");

exit(1);

}

if (gotstart)

{

printf("More than one entry point\n");

exit(1);

}

gotstart = 1; // indicate S entry processed

start = addr + mcaindex; // save adjusted address

}

else

if (letter == 'G')

{

if (fread(&addr, 2, 1, infile) != 1)

{

printf("Invalid G entry\n");

exit(1);

}

Gadd[Gindex] = addr + mcaindex; // save adjusted address

j = 0;

do // get string in G entry

{

letter = fgetc(infile);

buf[j++] = letter;

} while (letter != '\0');

j = 0;

while (j < Gindex) // check for multiple definitions

{

if (!strcmp(buf, Gptr[j]))

{

printf("Multiple defs of global var %s\n", buf);

exit(1);

}

else

j++;

}

Gptr[Gindex++] = strdup(buf); // save string

}

else

if (letter == 'E')

{

*code missing here*

}

else

if (letter == 'e')

{

*code missing here*

}

else

if (letter == 'V')

{

*code missing here*

}

else

if (letter == 'A')

{

*code missing here*

}

else

{

printf("Invalid header entry in %s\n", argv[i]);

exit(1);

}

}

// add machine code to machine code array

while(fread(&inst, 2, 1, infile))

{

mca[mcaindex++] = inst;

}

fclose(infile);

}

//================================================================

// Step 2: Adjust external references

// handle E references

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

{

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

if(!strcmp(Eptr[i], Gptr[j]))

break;

if (j >= Gindex)

{

printf("%s is an undefined external reference", Eptr[i]);

exit(1);

}

mca[Eadd[i]] = (mca[Eadd[i]] & 0xf800) |

((mca[Eadd[i]] + Gadd[j] - Eadd[i] - 1) & 0x7ff);

}

// handle e entries

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

{

*code missing here*

}

// handle V entries

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

{

*code missing here*

}

//================================================================

// Step 3: Handle A entries

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

*Code missing here. Only 1 statement needed to handle each A entry*

//================================================================

// Step 4: Write out executable file

outfile = fopen("link.e", "wb");

if (!outfile)

{

printf("Cannot open output file link.e\n");

exit(1);

}

printf("Creating executable file link.e\n");

// Write out start entry if there is one

if (gotstart)

{

fwrite("S", 1, 1, outfile);

fwrite(&start, 2, 1, outfile);

}

// Write out G entries

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

{

fwrite("G", 1, 1, outfile);

fwrite(Gadd + i, 2, 1, outfile);

fprintf(outfile, "%s", Gptr[i]);

fwrite("", 1, 1, outfile);

}

// Write out V entries as A entries

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

{

*Code missing here:*

*Write out V entries as A entries*

}

// Write out A entries

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

{

*Code missing here:*

*Write out A entries*

}

// Terminate header

fwrite("C", 1, 1, outfile);

// Write out code

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

{

fwrite(mca + i, 2, 1, outfile);

}

fclose(outfile);

}