

CIS 2107
Computer Systems and Low-Level Programming
Fall 2010
Final Exam

December 16, 2010

Name: _____

Page	Points	Score
1	10	
2	15	
2	11	
4	8	
5	8	
6	23	
8	15	
10	10	
Total:	100	

Instructions

The exam is closed book, closed notes. You may *not* use a calculator, cell phone, etc.

For each of the questions, unless otherwise specified, you can assume the following sizes for C data types:

type	bytes
char	1
short	2
int	4
long	8
float	4
double	8
void*	4

```

=====
CIS 2107 Summary Card 28-jan-08
=====

=====
Unix
=====
Compiling and executing a C program

unix> nano hello
#include <stdio.h>
int main()
{
    printf("Hello world\n");
}
^x
unix> gcc hello.cn
unix> ./a.out
Hello world
unix>

-----
unix>hexdump -C hello.c
00000000 23 69 6e 63 6c 75 64 65 20 3c
       73 74 64 69 6f 2e  |#include <stdio.|
00000010 68 3e 0a 69 6e 74 20 6d 61 69
       6e 28 29 0a 7b 0a  |h>.int main()|.|
00000020 20 20 20 20 20 70 72 69 6e 74 66
       28 22 48 65 6c 6c  | printf("Hell|
00000030 6f 20 44 72 2e 20 42 6f 62 20
       5c 6e 22 29 3b 0a  |o Dr. Bob \n");.|
00000040 7d 0a          |}|
unix>

-----
unix> gcc -E -o hello.i hello.c
unix> gcc -S -o hello.s hello.i
unix> gcc -c -o hello.o hello.s
unix> gcc -o hello hello.o
unix> ./hello
Hello world
unix>

-----
+-----+ +-----+
hello.c| pre- |hello.i| |
----->|processor|----->|Compiler|-->|
      | (cpp) | | (ccl) | | |
      +-----+ +-----+ |
                                           v
|<-----|
| |
| | +-----+ +-----+
V hello.s| |hello.o| |hello
----->|Assembler|----->|linker|---->
      | (as) | | (ld) | |
      +-----+ +-----+

-----
gcc commands

gcc -E -o prog.i prog.c preprocess only
gcc -S -o prog.s prog.c plus compile
gcc -c -o prog.o prog.c plus assemble
gcc -o prog prog.c plus link
gcc -v prog.c see detail
gcc -o prog one.c two.s three.o
-----

```

```

-----
Other Commands

hexdump -C file.c      hex dump
objdump -d prog.o      disassemble .o
objdump -d prog        disassemble exe
objdump -x prog.o      header contents
gdb prog               debug prog
x/20b main             examine 20 bytes of main

For help with any of the following, use
"cmd --help" or "man cmd" or "info cmd"
as, cpp, gcc, gdb, help, info,
ld, man, objdump, od
-----

```

```

=====
C Programming Language
=====
Identifiers

```

1. Letters, numbers, underscores "_".
2. Begin with a letter (library routines use names beginning with "_").
3. Keywords reserved.
4. Declaration - interpretation of identifier, many times.
5. Definition - declares and reserves storage, only once.

keywords

auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while

type specifiers

char	int	unsigned	union
double	long	struct	unsigned
enum	short	typedef	void
float			

Basic Data Types on x86 unix

1. char - 8-bit signed int, -128 to 127. ASCII character from 0 to 127.
2. short - 16-bit signed integer.
3. long - 32-bit signed integer.
4. int - 32-bit signed integer.
5. long long - 64-bit signed integer
6. float - 32-bit floating point number.
7. double - 64-bit floating point.
8. long double - 80-bit floating point.
9. void - nonexistent value

Data Type Modifiers

1. unsigned or signed (default)
2. const (never changed)
3. volatile (don't use optimization)

```

-----
Storage Classes and Linkage of Objects
(An object is a named region of storage)

```

1. Internal (to a block or function)
 - a. automatic - known only in block after def. No init. Discard on exit. (int i; auto int i;).
 - b. static - known only in block after def. Init to zero. Preserved on exit. (static int i;)
2. External (to all blocks).
 - a. external linkage (default). Known in file after def (1) or anywhere after extern (many). Init to zero. (int i; extern int i;)
 - b. internal linkage using static keyword - known in file below def. or in file after extern. Init to zero. (static int i;)

Derived Types

1. arrays of objects of a given type
2. functions returning objects of type
3. pointers to objects of a given type
4. structures containing a sequence of objects of various types
5. unions containing any one of several objects of various types

Declaration examples.

```

int var;           32-bit signed int
int *var;          pointer to int
int var[3];        array of 3 ints
int *var[3];       array of 3 ptrs to ints
int *(var[3]);     array of 3 ptrs to ints
int (*var)[3];     ptr to array of 3 ints
int f(void);       function returns int
int *f(void);      function returns ptr
                  to int
int (*p)(void);    p is ptr to function
                  that returns int
void f(int *a);    arg to f is ptr to int
void f(int *);     same - "a" is a comment

```

```

int x[2][3][4]; /*definition*/
x[1][1][1] is int
x[1][1] is array of 4 ints
x[1] is a 3x4 array of ints
x is a 2x3x4 array if ints
x[1][1], x[1], and x are pointers
-----

```

Type names - declare an object and omit the object name. Used in casts, function declarations, and sizeof

type name	declaration
int	int var;
int *	int *var;
int *[3]	int *var[3];
int (*)[]	int (*var)[];
int (int)	int fun(int);
int *(void)	int *fun(void);
int (*)(void)	int (*var[]) (void);

```

-----
Signed type conversions

char -> short -> long-> float -> dou
-----
Integer constants (suffix u, U, l, L)
123    32-bit int
123UL  32-bit unsigned long
0100    100 octal is 64 decimal
0x100   100 hexadecimal is 256 deci
-----
Floating constants (suffix f, F, l, L)
12.3    64-bit double
12.3F   32-bit float
12.3L   80-bit long double
123e-01 == 12.3
-----
Character and String Constants
'x'      8-bit character constant
'a' == '\101' == 'x41' == 61
"string" array of 7 char ('\0' ac
-----

```

```

Enumeration Constants
enum day {sun=1, mon, tues, wed}
-----
C Operators in decreasing precedence

```

	() (grp)	[] (sub)	
struct ->	(mbr)	.	(mbr)
unary	! (not)	~ (1's)	++ (inc)
unary	-- (dec)	+ (pos)	- (neg)
unary	* (ind)	& (adr)	
unary	(type)	sizeof	
arith	/ (div)	* (mul)	% (mod)
arith	+	(add)	- (sub)
shift	<< (lft)	>> (rgt)	
relat	< (ls)	<= (le)	> (gt)
relat	>= (ge)	== (eq)	!= (ne)
bit	& (and)	^ (exor)	(or)
logic	&& (and)	(or)	
if	? (true)	:	(false)
assign	=	+=	-=
assign	*	r-l	/=
assign	%	&=	^=
assign	=	<<=	>>=
stmt	,		

Operators

1. struct.member, union.member
2. ptr_to_struct->member
3. ptr_to_union->member
3. struct->member <==> (*struct).member
4. *&var <==> var
5. expl<<exp2 <==> expl*(2**exp2) if no overflow
6. expl>>exp2 <==> expl/(2**exp2) if positive and no overflow
7. for negative expl, expl>>exp2 may fill with 0 (unsigned) or 1 (sig)

Typical Program Parts

```

#include ...;
#define ...;
function prototypes;
external declarations and definitio
int main(argc, *argv[] {...}
ret-type funct-name(arg-decl) {...}
ret-type funct-name(arg-decl) {...}
-----

```

-----page 2
Typical function parts (including main)

```
ret-type funct-name(arg-decl)
{
    internal definitions and decl.;
    statement;
    statement;
    return expression;
}
```

Composition of Declarations & Statements

tokens: identifiers, keywords, constants
string literals, operators, other
white space: sp, ht, vt, nl, ff, comment
declaration: legal arrangement of tokens
statement: legal arrangement of tokens

Types of Statements

labeled-stmt.	label: stmt;
expression-stmt.	expression;
compound-statement.	{stmt; stmt; stmt;}
selection-stmt.	if, switch
iteration-stmt.	while, do, for
jump-stmt.	goto, continue, break, return,

while, for, do, if

```
while (exp) {...}
for (exp; exp; exp) {...}
do {...} while (exp);
if (expression) {...} else {...}
exp ? nonzerostmt : zerostmt;
```

switch

```
switch (intexp) {
    case int1: stmts;
    case int2: stmts; break;
    default: stmts;
}
```

Jump statements and statement labels

```
if (exp) goto error;
...
error: stmts;
```

```
continue; /*end this iter; start next*/
break; /*end iteration or switch */
return exp; /*return to stmt after call*/
```

C Preprocessor

```
#include "filename" /* local dir */
#include <filename> /* sys defined */
#define name replacement text
#define forever for (;;) {}
#define max(A,B) ((A) > (B) ? (A) : (B))
/* max(i++, j++); /* bad idea */
#if #ifdef #ifndef #elif #else #endif

#if !defined(ABC)
#define ABC
#else
#define ABC
#endif
#endif
```

typedef - new names for old data type

```
typedef int Length; Length a, b, c;
void (*signal(int signum,
               void (*func)(int)))(int);
typedef void Sigfun(int);
Sigfun *signal(int signum, Sigfun *fun);
```

Structures and Unions

```
struct OptStructName {
    declaration1;
    declaration2;
} OptVar1, OptVar2;
struct StructName Var3, Var4;
```

```
struct point {
    int x;
    int y;
} xy, *pxy
xy.x = 5; xy.y=10;
pxy = &xy;
pxy->x = 20; pxy->y = 30;
```

```
typedef struct Flt { /*little endian*/
    unsigned int fract:23;
    unsigned int exp:8;
    unsigned int sign:1;
} flt;
union Tstflt {
    flt also3;
    float f;
} tstflt;
flt a3 = {0x400000,128,0};
testflt.also3=a3;
print("%12.10f\n", tstflt.f);
```

Binary Trees

```
struct tnode {
    int data;
    struct tnode *left;
    struct tnode *right;
}
```

```
struct tnode *node = (struct tnode *)
    malloc(sizeof(struct tnode));
node->data = 1;
node->left = node->right = null;
```

```
typedef struct tnode Treenode;
typedef struct tnode *Treeptr;
Treeptr node = (Treeptr)
    malloc(sizeof(Treenode));
```

Linked Lists

```
struct llist {
    struct llist *next;
    int value;
}
for (p = head; p != NULL; p = p->next)
-----
Printf format specifiers (after %)
```

i,d	ints	x	hex	e,f,g	double
u	uint	c	char	p	pointer
o	octal	s	string		

Examples

```
for (i=1, j=n; i>n; ++i, --j) {...}
```

```
struct tnode *node = (struct tnode *)
    malloc(sizeof(struct tnode));
```

```
for (p = head; p != NULL; p = p->next)
```

```
int factorial(int i) {
    return i<=1 ? 1 : i*factorial(i-1);
}
```

ANSI C Standard Library

Input and Output <stdio.h>

```
FILE *f, char *s, *t, int c, i, j
```

f=fopen(s, t)	i=fclose(f)
i=fopen(f, s, ...)	i=printf(s, ...)
i=sprintf(s, t, ...)	i=scanf(f, s, ...)
i=scanf(s, ...)	i=sscanf(s, t, ...)
i=fgetc(f)	t=fgets(s, i, f)
i=fputc(c, f)	i=fputs(s, f)
i=getc(f)	i=getchar(void)
t=gets(s)	i=putc(j, f)
i=putchar(j)	i=puts(s)
i=ungetc(j, f)	

Character Class Tests <ctype.h>

```
int i, char c
```

i=isalnum(c)	i=isalpha(c)	i=iscntrl(c)
i=isdigit(c)	i=isgraph(c)	i=islower(c)
i=isprint(c)	i=isspace(c)	i=issupper(c)

string functions (incomplete) <string.h>

```
char *s, *t, const char *cs, *ct
size_t n, int c, i
```

s=strcpy(s,ct)	s=strncpy(s,ct,n)
s=strcat(s,ct)	s=strncat(s,ct,n)
i=strcmp(cs,ct)	i=strncmp(cs,ct,n)
s=strchr(ct,c)	s=strrchr(ct,c)
s=strstr(cs,ct)	n=strlen(cs)
s=strerror(n)	

Mathematical Functions (math.h)

double x, y, z, int n		
z=sin(x)	z=cos(x)	z=tan(x)
z=asin(x)	z=acos(x)	z=atan(x)
z=atan2(y,x)	z=sinh(x)	z=cosh(x)
z=tanh(x)	z=exp(x)	z=log(x)
z=log10(x)	z=pow(x,y)	z=sqrt(x)
z=ceil(x)	z=floor(x)	z=fabx(x)

Utility Functions <stdlib.h>

```
double atof(const char *s)
int atoi(const char *s)
long atol(const char *s)
int rand(void)
void stand(unsigned int seed)
void *calloc(size_t nobj, size_t size)
void *malloc(size_t size)
void free(void *p)
void exit(int status)
int abs(int n)
int labs(long n)
```

=====

as Assembler (info as)

AT&T vs. Intel Syntax

```
.intel_syntax noprefix
.att_syntax prefix
```

AT&T	Intel
------	-------

push \$4	push 4
clr %eax	clr eax
jump *%eax	jump eax
addl \$4, %eax	add eax, 4
movb movw	mov (and examine
movl movq	operands)
movb \$10,achar	mov byte ptr achar

AT&T addr. SEC:DISP(BASE,INDEX,SCALE)
Intel addr. SEC:(BASE+INDEX*SCALE+DISP)

Assembler (as) Syntax

1. /* this is a "multiline" comment
2. # this is a "line" comment
3. Statements end at '\n' or ';'.
4. Symbols (A-Z), (a-z), (0-9), (._\$)
5. symbol: - define symbolic address
6. .symbol - assembly directive
7. symbol - op code or address

Assembler Operands

zero, one, and two operand instructions

setc	# set the carry bit
incl wage	# increment long word
movb \$5,x(%eax)	# x[%eax] = 5;

operands

1000	- mem. addr. 1000 contains constant
beta	- addr. beta contains operand
\$55	- immediate operand 55
%eax	- 32-bit eax reg. contains constant
(%eax)	- eax contains address of operand
beta(%eax,%ebx,4)	- address beta + %eax + 4*ebx contains operand

section - block of addresses at 0000

as predefined sections text, data, bss

.text	contains code
.data	initialized data (int, float, double)
.bss	uninitialized data (int, float, double)
.section .rodata	
.section .note.GNU-stack,"",@progbits	

other sections

```
undefined value is 0, ld will default to 0
.comment .ident text goes here
as address=(SECTION)+(OFFSET INTO SECTION)
.comm defines symbol in .bss section
```

symbols

```
symbol=expression #same as .set
.set symbol expression #same as =
.Lsymbol is local symbol (ld never sees it)
1:, 2:, 3: are local labels, 1f, 2b, 3b
"." is location counter. myadr: .label
```

Convert (sign extend) instructions

Classic Two Operand (source+dest->dest)
(add, adc, and, xor, or, sbb, sub, c

```
addX %reg,%reg      adcX %reg,%reg
addX $imm,%reg      adcX $imm,%reg
addX mem,%reg       adcX mem,%reg
addX %reg,mem       adcX %reg,mem
addX $imm,mem       adcX $imm,mem
```

subX %reg,%reg	sbbX %reg,%reg
subX \$imm,%reg	sbbX \$imm,%reg
subX mem,%reg	sbbX mem,%reg
subX %reg,mem	sbbX %reg,mem
subX \$imm,mem	sbbX \$imm,mem

andX %reg,%reg	orX %reg,%reg
andX \$imm,%reg	orX \$imm,%reg
andX mem,%reg	orX mem,%reg
andX %reg,mem	orX %reg,mem
andX \$imm,mem	orX \$imm,mem

```
xorX %reg,%reg      cmpX %reg,%reg
xorX $imm,%reg      cmpX $imm,%reg
xorX mem,%reg       cmpX mem,%reg
xorX %reg,mem        cmpX %reg,mem
xorX $imm,mem        cmpX $imm,mem
```

```
testX %reg,%reg      testX $imm,%reg
-----
Classic 1 Operand (op dest->dest)
```

```
incX %reg      decX %reg
incX mem       decX mem
```

```
notX %reg      negX %reg
notX mem       negX mem
```

Shift and Rotate Instructions
(rol, ror, rcl, rcr, shl=sal, shr, sll, srl, swp)
If no \$imm, shift amount in %cl

```
salX $imm,%reg      sarX $imm,%reg
salX $imm,mem        sarX $imm,mem
```

```
shlX $imm,%reg      shrX $imm,%reg
shlX $imm,mem        shrX $imm,mem
```

```
rolX $imm,%reg      rorX $imm,%reg
rolX $imm,mem        rorX $imm,mem
```

```

rclX $imm,%reg      rcrX $imm,%reg
rclX $imm,mem        rcrX $imm,mem

```

Unsigned Multiply and Divide
oper = %reg or mem
for divide, result is remainder, quotient

```

mulb oper    %ax    = %al * oper
mulw oper    %dx,%ax = %ax * oper
mull oper    %edx,%eax = %eax * oper
divb oper    %ah,%al = %ax / oper
divw oper    %dx,%ax = %dx,%ax / op
divl oper    %edx,%eax = %edx,%eax /

```

Signed Multiply and Divide
oper = %reg or mem
for divide, result is remainder,quotient

```
imulb oper    %ax      = %al * oper
imulw oper    %dx,%ax  = %ax * oper
imull oper    %edx,%eax = %eax * oper
```

```
imulX %reg,%reg
imulX %mem,%reg
imulX %imm,%reg
```

```
idivb oper    %ah,%al  = %ax / oper
idivw oper    %dx,%ax  = %dx,%ax / oper
idivl oper    %edx,%eax = %edx,%eax / oper
```

Jump, Call, and Ret Instructions

```
jmp label      #uses %eip relative addr.
jmp *%reg       #jump to addr in reg
jmp *mem        #jump to addr in mem loc
```

```
call pushes %eip (return addr) on stack
call label      #uses %eip relative addr.
call *%reg       #function addr in reg
call *mem        #function addr in mem loc
ret              #pop stack into %eip
```

Condition Codes

CF: Carry flag (unsigned overflow if 1)
ZF: Zero flag, set to 1 if result is 0
SF: Sign flag, equals result high bit
OF: Overflow flag (unsigned overflow)

Conditional Jump instructions, jCC label
Use after cmp, add, sub, etc. (not mov)
cmpl a,b; jg label; jumps if b-a>0 (b>a)

Simple Condition Code Tests

```
jo label      jno label
jz label      jnz label
js label      jns label
jo label      jn0 label
```

signed unsigned

```
jg (jnl) label   ja (jnb) label
jge (jnl) label  jae (jnb) label
jl (jnge) label  jbe (jna) label
jle (jng) label  jbe (jna) label
```

Set Instructions (setCC reg8)

```
setCC reg8; movzbl reg8,reg32;
```

Set Individual Condition Code Bits

```
seto label      setno label
setz label      setnz label
sets label      setns label
seto label      setno label
```

signed unsigned

```
setg (setnl) reg8  seta (setnbe) reg8
setge (setnl) reg8 setae (setnb) reg8
setl (setnge) reg8 setb (setnae) reg8
setle (setng) reg8 setbe (setna) reg8
```

Floating Point (8 80-bit registers)

```
float a, b, c;      double d, e, f;
a = b + c;          d = e + f;

flds b #s=short      fldl e #l = long
flds c #p=pop         fldl f #p = pop
faddp %st,%st(1)      faddp %st,%st(1)
fstps a               fstpl d
```

Floating Point Condition Codes. After
"fnstsw %ax; andb \$69,%ah", %ah = (\$0,
\$1, \$64 or \$69) for (>, <, =, or NAN)

String Instructions

```
%esi points to source, %edi to dest.
%esi and %edi incremented by 1, 2, or 4
if DF (direction flag) == 0 (cld)
decrement if DF == 1 (std)

movs move string,      movX (%esi),(%edi)
cmps compare string,   cmpX (%esi),(%edi)
stos store string,      movX %a?,(%edi)
lods load string,       movX (%edi),%a?
scas scan string,       cmpX (%edi),%a?
```

put count in %ecx and repeat instruction
using rep, repe, and repne prefix

Read Time Stamp Counter
rdtsc - cycles since boot in %edx, %ecx

Counting and Number Representation

Signed and unsigned 8-bit bytes

hex binary unsigned 2's comp

```
00 00000000      0      0
01 00000001      1      1
02 00000010      2      2
03 00000011      3      3
...
7E 01111110     126     126
7F 01111111     127     127
80 10000000     128    -128
81 10000001     129    -127
...
FE 11111110     254      -2
FF 11111111     255      -1
```

Signed and unsigned 32-bit integers

```
hex                    unsigned                    2's comp

00000000      0      0
00000001      1      1
00000002      2      2
00000003      3      3
...
7FFFFFFF     2,147,483,646  2,147,483,646
7FFFFFFF     2,147,483,647  2,147,483,647
80000000     2,147,483,648 -2,147,483,648
80000001     2,147,483,649 -2,147,483,647
...
FFFFFFFFD     4,294,967,294      -3
FFFFFFFFE     4,294,967,294      -2
FFFFFFFFF     4,294,967,295      -1
```

ASCII

```
10 16 ch   10 16 ch   10 16 ch   10 16 ch

0 00 \0    32 20 SP   64 40 @    96 60 `
1 01 SOH   33 21 !    65 41 A    97 61 a
2 02 STX   34 22 "    66 42 B    98 62 b
3 03 ETX   35 23 #    67 43 C    99 63 c
4 04 EOT   36 24 $    68 44 D   100 64 d
5 05 ENQ   37 25 %    69 45 E   101 65 e
6 06 ACK   38 26 &    70 46 F   102 66 f
7 07 BEL   39 27 '    71 47 G   103 67 g
8 08 BS    40 28 (    72 48 H   104 68 h
9 09 HT    41 29 )    73 49 I   105 69 i
10 0A \n   42 2A *    74 4A J   106 6A j
11 0B \v   43 2B +    75 4B K   107 6B k
12 0C \f   44 2C ,    76 4C L   108 6C l
13 0D \r   45 2D -    77 4D M   109 6D m
14 0E SO    46 2E .    78 4E N   110 6E n
15 0F SI    47 2F /    79 4F O   111 6F o
16 10 DLE  48 30 0    80 50 P   112 70 p
17 11 DC1   49 31 1    81 51 Q   113 71 q
18 12 DC2   50 32 2    82 52 R   114 72 r
19 13 DC3   51 33 3    83 53 S   115 73 s
20 14 DC4   52 34 4    84 54 T   116 74 t
21 15 NAK   53 35 5    85 55 U   117 75 u
22 16 SYN   54 36 6    86 56 V   118 76 v
23 17 ETB   55 37 7    87 57 W   119 77 w
24 18 CAN   56 38 8    88 58 X   120 78 x
25 19 EM    57 39 9    89 59 Y   121 79 y
26 1A SUB   58 3A :    90 5A Z   122 7A z
27 1B ESC   59 3B ;    91 5B [   123 7B {
28 1C FS    60 3C <    92 5C \   124 7C |
29 1D GS    61 3D =    93 5D ]   125 7D }
30 1E RS    62 3E >    94 5E ^   126 7E ~
31 1F US    63 3F ?    95 5F _   127 7F DE
```

Escape sequences in ASCII

```
null or zero   NUL \0 = \000 = \x00 = 0
audible alert  BEL \a = \007 = \x07 = 7
backspace      BS \b = \010 = \x08 = 8
horizontal tab  HT \t = \011 = \x09 = 9
newline (LF)   NL \n = \012 = \x0a = 10
vertical tab   VT \v = \013 = \x0b = 11
formfeed       FF \f = \014 = \x0c = 12
carriage ret   CR \r = \015 = \x0d = 13
double quote   " \" = \042 = \x22 = 34
single quote   ' \' = \047 = \x27 = 39
question mark  ? \? = \077 = \x3f = 63
backslash      \ \\ = \134 = \x5c = 92
octal number   ooo \ooo
hex number     hh \hhh
```

Powers of 2 and 16

```
n                    2^n                    16^n

0                    1                    1
1                    2                    16
2                    4                    256
3                    8                    4,096
4                    16                    65,536
5                    32                    1,048,576
6                    64                    16,777,216
7                    128                    268,435,456
8                    256                    4,294,967,296
9                    512                    68,719,476,736
10                   1,024                   1,099,511,627,776
```

Decimal-Binary-Hexadecimal (Hex)

```
10        2    16                    10        2

0    0000    0                    8    1000
1    0001    1                    9    1001
2    0010    2                    10   1010
3    0011    3                    11   1011
4    0100    4                    12   1100
5    0101    5                    13   1101
6    0110    6                    14   1110
7    0111    7                    15   1111
```

Floating Point Numbers

```
len sign exp hide f1

float                    32    1    8    1
double                   64    1    11    1

long double              80    1    15    -

exponent bias is 127, 1023, or 16383
```

```
31 30    23 22
+-----+-----+
float |s| exp |1. + 23-bit fract|
+-----+-----+
```

Recognizing Floating Point Numbers

```
if e==0:
    denormalized
    f=(-1)^S * 2^(1-bias)*0.M
else if e == all 1s:
    if M == 0:
        if S == 0:
            +infinity
        else:
            -infinity
    else:
        NaN
else:
    normalized
    (-1)^S * 2^(e-bias)*1.M
```

```
#define bit_get(p,m) ((p) & (m))
#define bit_set(p,m) ((p) |= (m))
#define bit_clr(p,m) ((p) &= ~(m))
#define bit_flip(p,m) ((p) ^= (m))
#define bit_write(c,p,m)
(c ? bit_set(p,m) : bit_clear(p,m))
#define BIT(x) (0x01 << (x))
#define LONGBIT(x)
((unsigned long)0x00000001 << (x))
```

```
struct one32bitlong {
    unsigned bit:1;
    unsigned nibble:4;
    unsigned dozen:12;
    unsigned fifteen:15;
};
```

I. Short Answer

- (A) **The compilation process.** At times during the semester, we've gone through each of the steps of the compilation process separately.

We did:

- `gcc -E flurg.c` to get `flurg.i`.
- `gcc -S flurg.i` to get `flurg.s`.
- `gcc -c flurg.s` to get `flurg.o`.
- `gcc -o flurg flurg.o` to get the executable file `flurg`.

- (1 point) i. Describe what's in `flurg.i`. What do we call the process of translating `flurg.c` into `flurg.i`?

- (1 point) ii. Describe what's in `flurg.s`. What do we call the process of translating `flurg.i` into `flurg.s`?

- (1 point) iii. Describe what's in `flurg.o`. What do we call the process of translating `flurg.s` into `flurg.o`?

- (1 point) iv. Describe what's in `flurg`. What do we call the process of translating `flurg.o` and any necessary libraries into `flurg`?

- (1 point) v. Which component (type of program) decides in which register each C variable will be stored?

v. _____

- (1 point) (B) We've described the storage hierarchy in modern computers as a type of pyramid. Name two things that are generally true the higher up the pyramid we go.

- (2 points) (C) Other than the problem of differing libraries, explain why it is that I can't take a Windows executable, and run it on an Intel Mac or a machine running Linux on Intel. Please do not write, "because they're different operating systems". Be more specific about what the major problems are.

- (1 point) (D) Where do we store an integer return value from a function in x86 assembly?

- (1 point) (E) Memory allocated with `malloc()` is stored in what memory segment?

II. Data Representation

- (A) **Some bit operations** If we have `char i = 0x6B, j = 0xA7;`, what is the result of the following operations? Your answer must be in the form of exactly two hex digits¹.

(2 points)

i. `~i`

i. _____

(2 points)

ii. `!!i`

ii. _____

(2 points)

iii. `i|j`

iii. _____

(2 points)

iv. `i < (j & 0x0F)`

iv. _____

(2 points)

v. `i << 2`

v. _____

(1 point)

- (B) In hex, what is the smallest integer that can be represented by a 16-bit two's complement int?

(B) _____

(1 point)

- (C) In hex, what is the largest integer that can be represented by a 16-bit two's complement int?

(C) _____

(1 point)

- (D) In hex, what is the largest integer that can be represented by a 16-bit unsigned int?

(D) _____

(1 point)

- (E) In hex, what is -1 as an 16-bit two's complement int?

(E) _____

(1 point)

- (F) What is $110110111_2 + 11100101_2$ in base 2?

$$\begin{array}{r} 1 \quad 1 \quad 0 \quad 1 \quad 1 \quad 0 \quad 1 \quad 1 \quad 1_2 \\ + \quad \quad 1 \quad 1 \quad 1 \quad 0 \quad 0 \quad 1 \quad 0 \quad 1_2 \\ \hline \end{array}$$

(1 point)

- (G) What is $A59C_{16} + 2B8D_{16}$ in base 16?

$$\begin{array}{r} A \quad 5 \quad 9 \quad C \quad 2_{16} \\ + \quad \quad 2 \quad B \quad 8 \quad D_{16} \\ \hline \end{array}$$

¹Forget about the possibility of the values being promoted to 32-bits. Just behave as though we're living in the land of 8-bit arithmetic.

- (4 points) (H) Do one of the two following floating-point questions. (If you do both, we're going to grade the first one and ignore the second.)
- How would the number 6.25_{10} be stored in a 32-bit C float variable?

- Suppose that we have 0xC0400000 stored in a 32-bit C float variable.
 - Is the number positive or negative?

1) _____

- What is the bias?

2) _____

- What are the bits of the exponent part?

- What are the bits of the mantissa part?

- What floating-point number does this represent?

III. Some tricky declarations

Write a very brief description in English of what is declared. For example, if the question is `int func(int A[])`, you'd write, "func is a function which is passed an array of int and returns an int".

- (2 points) (A) `void (*p)(int);`

- (2 points) (B) `int (*p[])()`

- (2 points) (C) `int *(*p[5])()`

IV. Print Me

What is the value of each of the following after `func()` is called?

```
1  #include <string.h>
2  #include <stdio.h>
3  #include <stdlib.h>
4
5  typedef struct {
6      int x;
7      int *p;
8      char *s;
9  } Stuff;
10
11 void func(int [], Stuff);
12
13 int main(void)
14 {
15     int A[5]={10,20,30,40,50};
16     int x=60;
17     Stuff s;
18
19     s.x=70;
20     s.p=&x;
21     s.s=malloc(20);
22     strcpy(s.s, "long exam?");
23
24     func(A, s);
25
26     return 0;
27 }
28
29 void func(int A[], Stuff s)
30 {
31     s.x=777;
32     *(s.p)=666;
33     strcpy(s.s, "a little long");
34     s.s = malloc(20);
35     strcpy(s.s, "not too long");
36     A[0]=111;
37 }
```

(2 points) (A) `A[0]`

(A) _____

(2 points) (B) `s.x`

(B) _____

(2 points) (C) `*(s.p)`

(C) _____

(2 points) (D) `s.s`

(D) _____

pointers

V. If I have the following:

```
int main(void)
{
    int a=10;
    int b=20;
    int *p=&a;
    char *cp=(char*)&a;

    ...
    (*p)++;
    cp+=4;
    p+=2;
}
```

and memory is laid out like this:

<i>a</i>	1000	<input type="text"/>
<i>b</i>	1004	<input type="text"/>
<i>p</i>	1008	<input type="text"/>
<i>cp</i>	1012	<input type="text"/>

what do you see if you print:

(1 point)

(A) p

(A) _____

(1 point)

(B) *p

(B) _____

(1 point)

(C) &p

(C) _____

(1 point)

(D) cp

(D) _____

(1 point)

(E) (int)(*cp)

(E) _____

(1 point)

(F) &cp

(F) _____

VI. What is the value of each of the following assuming that the array **A** begins at address 1000 and **ints** are 4 bytes?

```
typedef struct {
    int x;
    int y;
    char s[12];
} Stuff;
```

```
int A[500];
```

```
int *ip=&A[0];
char *cp=&A[0];
Stuff *sp=&A[0];
```

```
ip+=10;
cp+=3;
sp+=5;
```

(1 point) (A) ip

(A) _____

(1 point) (B) cp

(B) _____

(1 point) (C) sp

(C) _____

(5 points)VII. For each of the following, suppose that **%eax** contains the value *x*, **%ecx** contains *y*. What's stored in **%edx** after the each operation?

expression	result
leal 0xC(%eax), %edx	
leal (%eax,%ecx), %edx	
leal (%eax,%ecx, 4), %edx	
leal 5(%eax,%eax,8), %edx	
leal 0xA(,%ecx,8), %edx	

(10 points) VIII. Write a C function equivalent to the following assembly (no credit for an answer containing inline assembly).

```
1      .text
2      .globl mystery
3      .type mystery, @function
4  mystery:
5      pushl %ebp
6      movl %esp, %ebp
7      movl 8(%ebp), %eax
8      addl $10, %eax
9      addl 12(%ebp), %eax
10     subl 16(%ebp), %eax
11     movl %ebp, %esp
12     popl %ebp
13     ret
```

(7 points) IX. Write a function which is passed an `int x` and returns the number 1 bits which would appear in `x`'s binary representation. Do not assume that an `int` is necessarily 4 bytes.

(5 points) X. Write a function which is passed a string `s`. The function removes the last character from `s`. For example, if `s` is `drinks`, after the function finishes, `s` is `drink`.

(10 points) XI. Write a function which is passed a pointer to a `List`, and an `int x`. The function appends `x` to the `List`. You may assume that the pointer to the `List` is not null. `List` and `list_node` are defined as follows:

```
struct list_node {  
    int data;  
    struct list_node *next;  
};  
  
typedef struct {  
    struct list_node *head;  
} List;
```

(10 points)XII. Write a function which is passed a sorted `int` array `A[]`, an `int len`, which is the length of `A[]`, and an `int x`. The function inserts `x` into `A[]` in its proper sorted position. For example, if `A[]` is `{10,30,40,50}`, and `x` is 20, after the function finishes, `A[]` is `{10,20,30,40,50}`. Do *not* use the `[]` operator.

```
void insert(int A[ ], int len, int x) {
```

(extra space)