mt. Code=66 Digipen login:_____

1. **Problem** (4 * 2 pts):

   Given these declarations, what is the expression below? Assume logical shift.

   ```
   unsigned short a = 7, b = 17, c = 35;
   ```

A) 1 B) 7 C) 14 D) 3 E) 16 F) none of the listed G) 17 H) 2 I) 5 J) 35 K) 15 L) 23	1-1._____ `b | a` 1-2._____ `c & b` 1-3._____ `c & b | a` 1-4._____ `c & b | a << 1`

2. **Problem** (6 pts):

   State the difference between logical and arithmetic shifts, provide a program that determines which shift is implemented by a compiler. Program should output either "L" or "A".

3. **Problem** (6 pts):

   One of the three most useful bit manipulating operations is setting a specific bit to 1.

   1. write code that sets n'th bit of integer i to 1, points are awarded for correctness and compactness:

   2. name the other 2 operations.

4. **Problem** (5 * 2 pts):

Convert C declaration into English

A) a pointer to a pointer to an array of 5 int

B) pointer to a function taking a pointer to function taking an array of 5 integers and returning an int and returning an int

C) a pointer to a pointer to a function that returns an int

D) a function that takes an int and returns a pointer to a pointer to an int

E) a pointer to an array of 5 pointers to int

F) an array of 5 pointers to functions taking an int and returning an int

G) a function that takes an int and returns a pointer to an array of 5 ints

H) a pointer to a function that takes an int and returns an int

I) legal declaration, but not in the list

J) a pointer to an array of 5 pointers to functions that return an int

K) pointer to a function taking a pointer to array of 5 ints and returning an int

L) function taking a pointer to array of 5 ints and returning a pointer to an int

M) an array of 5 pointers to pointers to int

N) illegal declaration

O) a pointer to an array of 5 pointers to functions taking an int and returning an int

P) a function that takes an int and returns a pointer to an array of 5 pointers to int.

4-1._____ `int (*foo[5])(int);`

4-2._____ `int *(foo[5])(int);`

4-3._____ `int *(*foo)[5];`

4-4._____ `int (*foo)(int (*)(int [5]));`

4-5._____ `int (*foo)(int(*)[5]);`

5. **Problem** (8 pts):

Write the C declaration for each of the English statements below. If a statement describes an illegal declaration, then write ILLEGAL.

a. foo is a function that takes an int and returns a pointer to an array of 5 pointers to int.

b. foo is an array of 5 pointers to functions taking an int and returning a pointer to int

c. foo is a function taking a pointer to (a function taking int and returning int) returning a pointer to an array of 2 ints

d. foo is a function taking a pointer to (a function taking int and returning int) returning an array of 2 ints

6. **Problem** (6 pts):

   Implement a `push_front` function that inserts a node at the front of a singly-linked list. Function should be able to return an error code to the caller, for example in the case of a memory allocation error.

7. **Problem** (6 pts):

   Adding or removing a node to the front of a singly-linked list is trivial since we always have a pointer to the first node (head). Adding to the end is a little more work because we have to walk the list to find the end. We do have a technique that allows us to find the end of the list quickly: use a tail pointer in addition to the head pointer. This makes it trivial to add to the end of the list. However, this efficient technique won't work very well when we have to remove the last node. Explain the problems with this "tail pointer" approach.

8. **Problem** (6 pts):

Complete the function below that returns a pointer to the last element in the list. Use the definition of a Node below. Return NULL if the list is empty. Points awarded for compactness of code.

```
struct Node {
  int number;
  struct Node *next;
};

struct Node* get_tail(struct Node *pHead){
```

9. **Problem** (10 * 1 pts):

For each of the following expressions determine whether it's legal.
For legal expression write down its value.
Assume non-cumulative execution, i.e. all modifications from previous lines ARE LOST.
Note that the value of assignment is value of the right-hand side, if the assignment is legal.

```
short array[] = {3, 6, 2, 4, 7, 8};
/* assume array = 1000, sizeof(int)=sizeof(pointer)=4, sizeof(short)=2 */
short *p1 = &array[1];
short *p3 = &array[3];
short *p5 = &array[5];
```

A) legal, but not listed B) 2 C) 3 D) -4 E) 5 F) 7 G) 8 H) 10 I) 1000 J) 1001 K) 1003 L) 1006 M) 1004 N) 1002 O) illegal P) 1008 Q) 1 R) 4 S) 6	9-1._____ `p5-p1` 9-2._____ `p1-(p3-p5)` 9-3._____ `p1-p3-p5` 9-4._____ `p1 += p5-p3` 9-5._____ `p1 -= p5-p3` 9-6._____ `p1 = p5 - 3` 9-7._____ `p5 - --p1` 9-8._____ `*--p3 = 2` 9-9._____ `*(p3++) = 5` 9-10._____ `p3+2 = &array[2]`

10. **Problem** (5 * 1 pts):

Given array declaration, what are the values in bytes?

```
short sha[]={1,2,3,4,5,6,7,8};
int array[8][4];
int (*ppi)[4] = array;
```

Assume sizeof(int)=4, sizeof(short)=2.

A) 4 B) 64 C) 1 D) 8 E) 2 F) 16 G) 32 H) 128	10-1._____ `sizeof(sha)` 10-2._____ `sizeof(array)` 10-3._____ `sizeof(array[2])` 10-4._____ `sizeof(ppi)` 10-5._____ `sizeof(ppi[2])`

11. **Problem** (6 pts):

Study the code below carefully. What does it print out? You should draw a diagram to help visualize the operations. (Hint: The outputted characters form a word.)

```
#include <stdio.h>

void main() {
  char *strs[] = {"the","exam","tones","curb",0};
  char** pps = strs;
  int i=0;

  while(*pps) {
    printf ("%c", *(*pps++ + i) + i);
    ++i;
  }
}
```

12. **Problem** (7 * 1 pts):

Given the following declaration,

```
int f1(void) { return 16;  }
int f2(void) { return 32;  }
int f3(void) { return 64;  }

int (*pf[])(void) = {f1, f2, f3};
```

evaluate expressions.

A) 32 B) 16 C) address of array D) address of f2 E) address of f1 F) address of f3 G) 64 H) illegal	12-1._____ `(*(pf+1))()` 12-2._____ `(*(pf+1))` 12-3._____ `(*pf+1)()` 12-4._____ `pf()` 12-5._____ `pf[1]()` 12-6._____ `(**(pf+1))()` 12-7._____ `f3`

13. **Problem** (6 pts):

Explain the difference between linear recursion and tail recursion.

Explain the meaning of expression "tail recursion removal".

14. **Problem** (6 pts):

Write a complete program (one that compiles without warnings or errors) that will simply print out all of the arguments that were specified on the command line. In other words, compiling the program to `foo.exe`, and then executing it on the command such as this:

```
foo.exe Mommy says "Eat your spinach!"
```

will cause this to be printed:

```
foo Mommy says Eat your spinach!
```

Points awarded for compactness of code.

15. **Problem** (6 pts):

The code below is in a header file named foo.h. Modify the header file so that it will be possible to use it with both C and C++ compilers. (6 points)

```
#ifndef FOO_H
#define FOO_H




int foo (char *infile);
int bar (double rate);




#endif
```

()	Grouping	exp	N/A
()	Function call	rexp	L-R
[]	Subscript	lexp	L-R
.	Structure member	lexp	L-R
->	Structure pointer member	lexp	L-R
++	Postfix increment	rexp	L-R
--	Postfix decrement	rexp	L-R
!	Logical negate	rexp	R-L
~	One's complement	rexp	R-L
+	Unary plus	rexp	R-L
-	Unary minus	rexp	R-L
++	Prefix increment	rexp	R-L
--	Prefix decrement	rexp	R-L
*	Indirection (dereference)	lexp	R-L
&	Address of	rexp	R-L
sizeof	Size in bytes	rexp	R-L
(type)	Type conversion (cast)	rexp	R-L
*	Multiplication	rexp	L-R
/	Division	rexp	L-R
%	Integer remainder (modulo)	rexp	L-R
+	Addition	rexp	L-R
-	Subtraction	rexp	L-R
<<	Left shift	rexp	L-R
>>	Right shift	rexp	L-R
>	Greater than	rexp	L-R
>=	Greater than or equal	rexp	L-R
<	Less than	rexp	L-R
<=	Less than or equal	rexp	L-R
==	Equal to	rexp	L-R
!=	Not equal to	rexp	L-R
&	Bitwise AND	rexp	L-R
^	Bitwise exclusive OR	rexp	L-R
|	Bitwise inclusive OR	rexp	L-R
&&	Logical AND	rexp	L-R
||	Logical OR	rexp	L-R
?:	Conditional	rexp	N/A
=	Assignment	rexp	R-L
+=	Add to	rexp	R-L
-=	Subtract from	rexp	R-L
*=	Multiply by	rexp	R-L
/=	Divide by	rexp	R-L
%=	Modulo by	rexp	R-L
<<=	Shift left by	rexp	R-L
>>=	Shift right by	rexp	R-L
&=	AND with	rexp	R-L
^=	Exclusive OR with	rexp	R-L
|=	Inclusive OR with	rexp	R-L
,	Comma	rexp	L-R